{Learn, Create, Innovate};

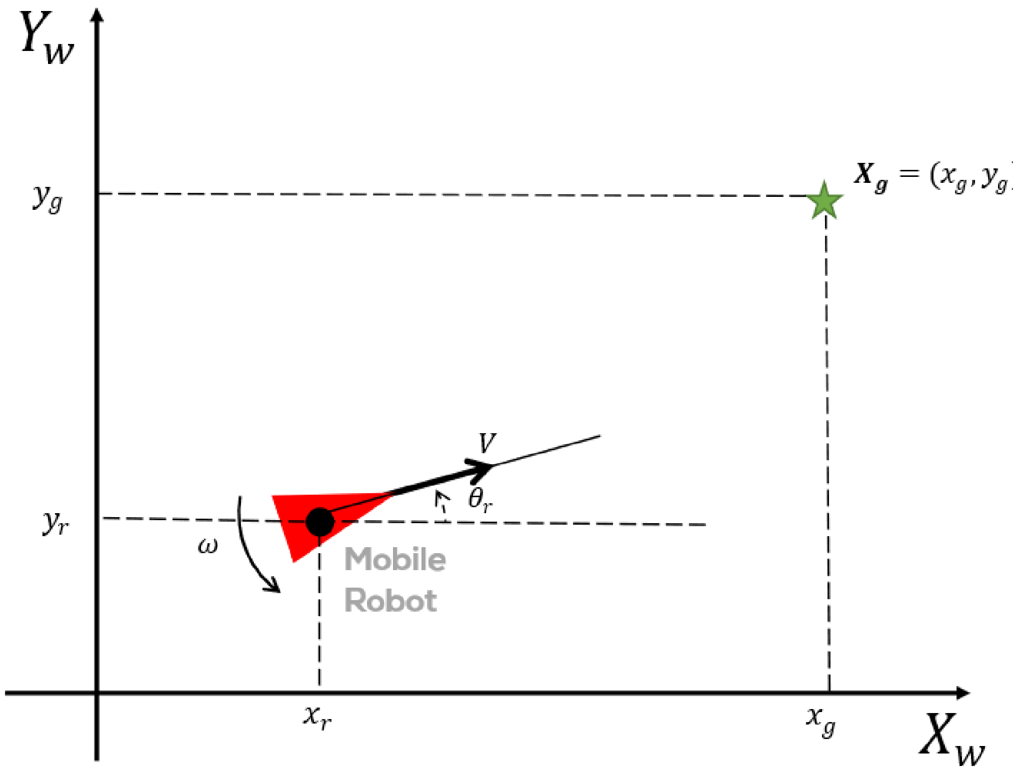
Autonomous Systems

Mobile Robot: Motion Control

Manchester Robotics

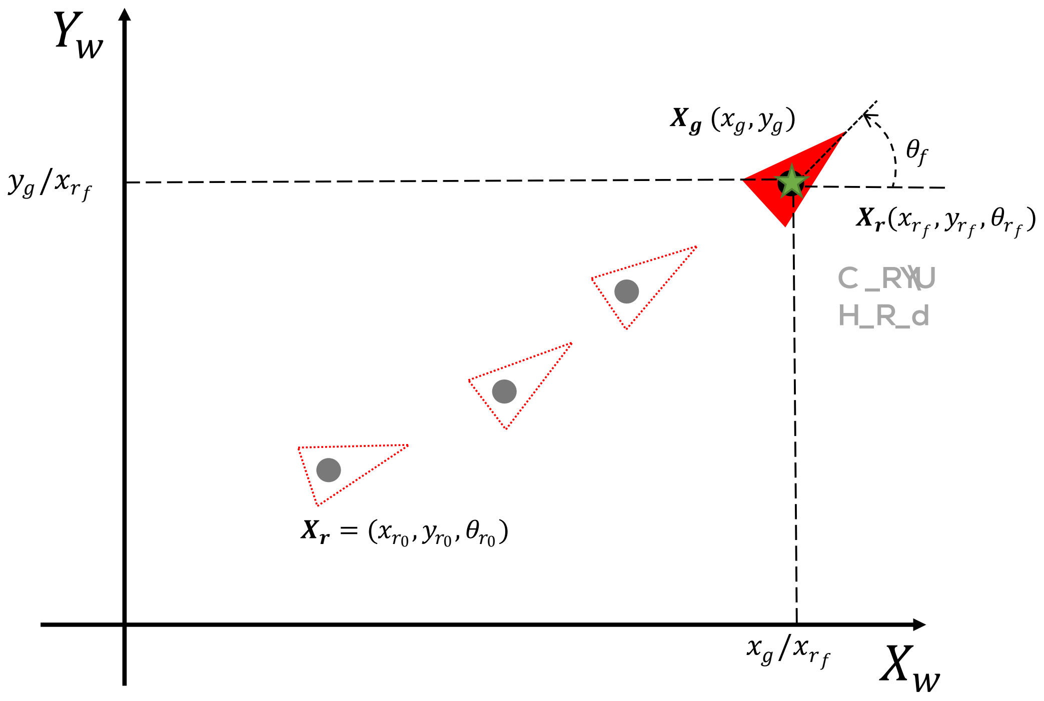
Motion Control

- The **motion control** for a mobile robot deals with the task of finding the control inputs that need to be applied to the robot such that a predefined goal can be reached in a finite amount of time.
- Control of differential drive robots has been studied from several points of view but essentially falls into one of the following three categories: **point-to-point navigation (or point stabilisation), trajectory tracking, and path following**.



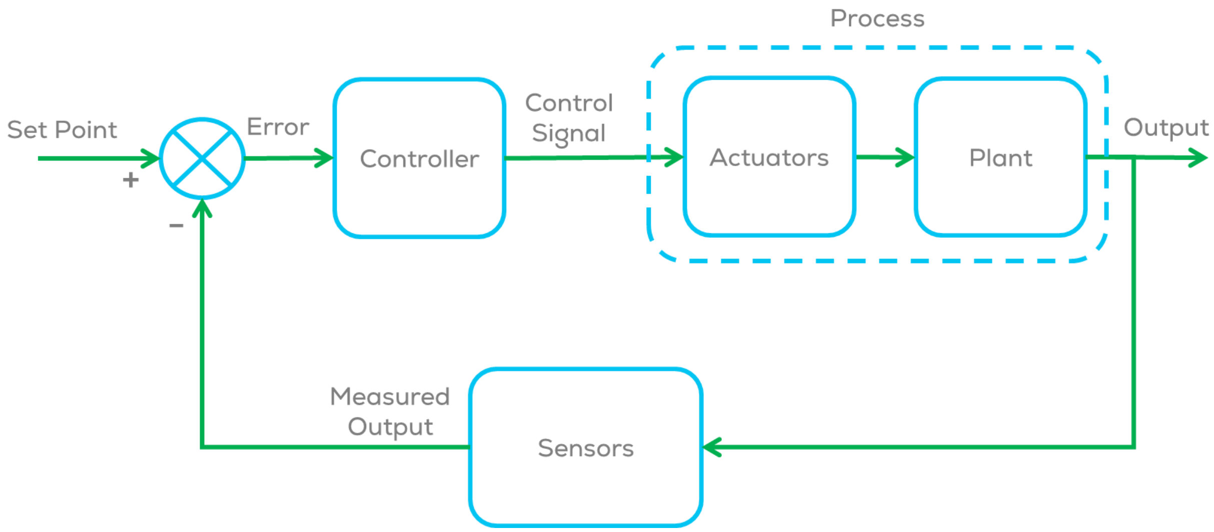
Motion Control: Point Stabilisation

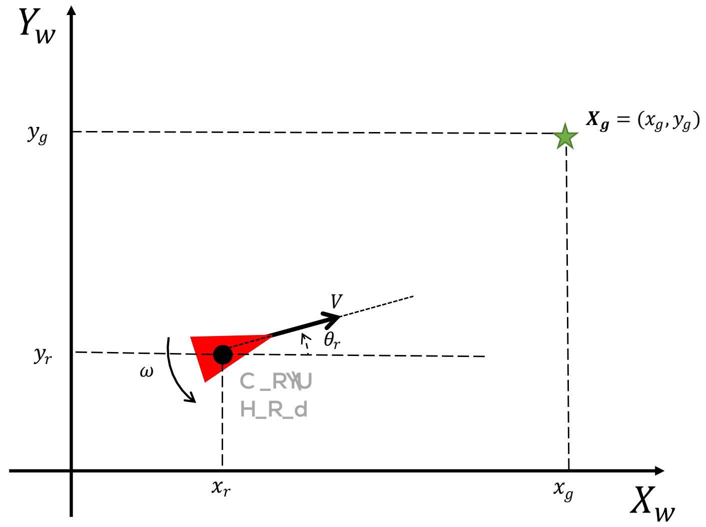
- The objective here is to drive the robot to a desired fixed state, say a fixed position and orientation.
- When the vehicle has nonholonomic constraints, point stabilisation presents a true challenge to control systems.
  - Since that goal cannot be achieved with smooth time-invariant state-feedback control laws.
- This control technique will be used in this course.



The Control System

- When Designing a Controller a question arises…
- How can we transform the classical feedback control diagram, to fit the purpose of controlling the position of a mobile robot using Point Stabilisation?
- … We start by defining the main things of the control diagram: Set Point, Measured Output, Control Signal, etc…

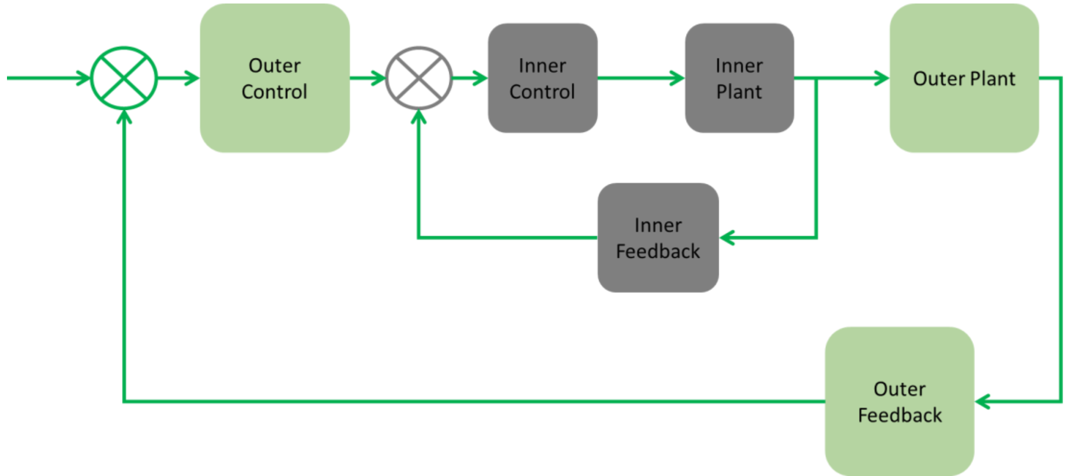


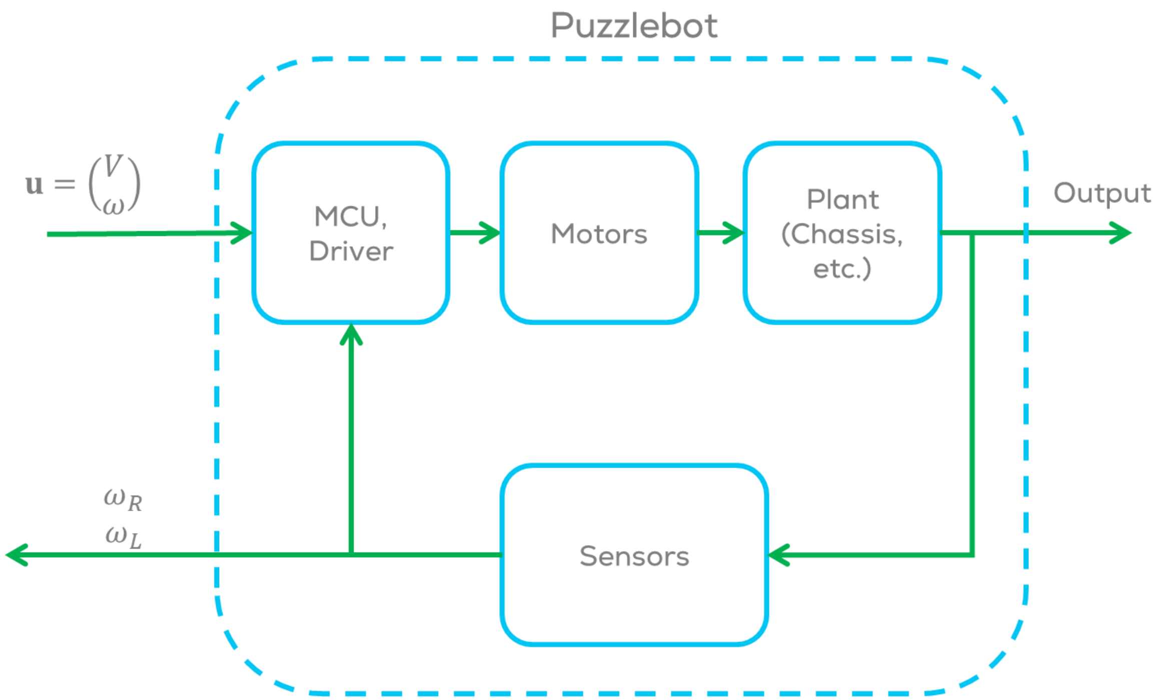


The Mobile Robot

Internal Controllers

- Mobile Robots usually contain internally some controllers to regulate the speed of the motors.
- These controllers must be taken into consideration when trying to control the robot using an external controller (cascade control)





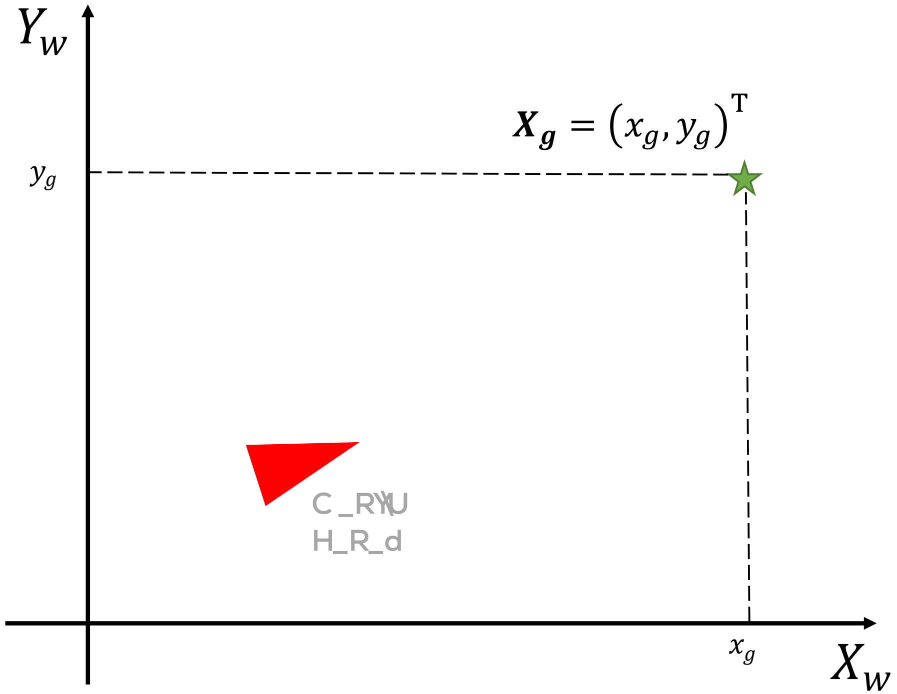
Point Stabilisation

Set Point

- The set point/goal can be defined in the simplest way as a set of coordinates in a multidimensional space. For instance, if the robot is moving in a two-dimensional space the goal is:

- $\boldsymbol{X}_g = \begin{pmatrix} x_g \\ y_g \end{pmatrix}$ if the position of the robot needs to be controlled, or

- $\boldsymbol{X}_g = \begin{pmatrix} x_g \\ y_g \\ \theta_g \end{pmatrix}$ if both position and heading need to be controlled.



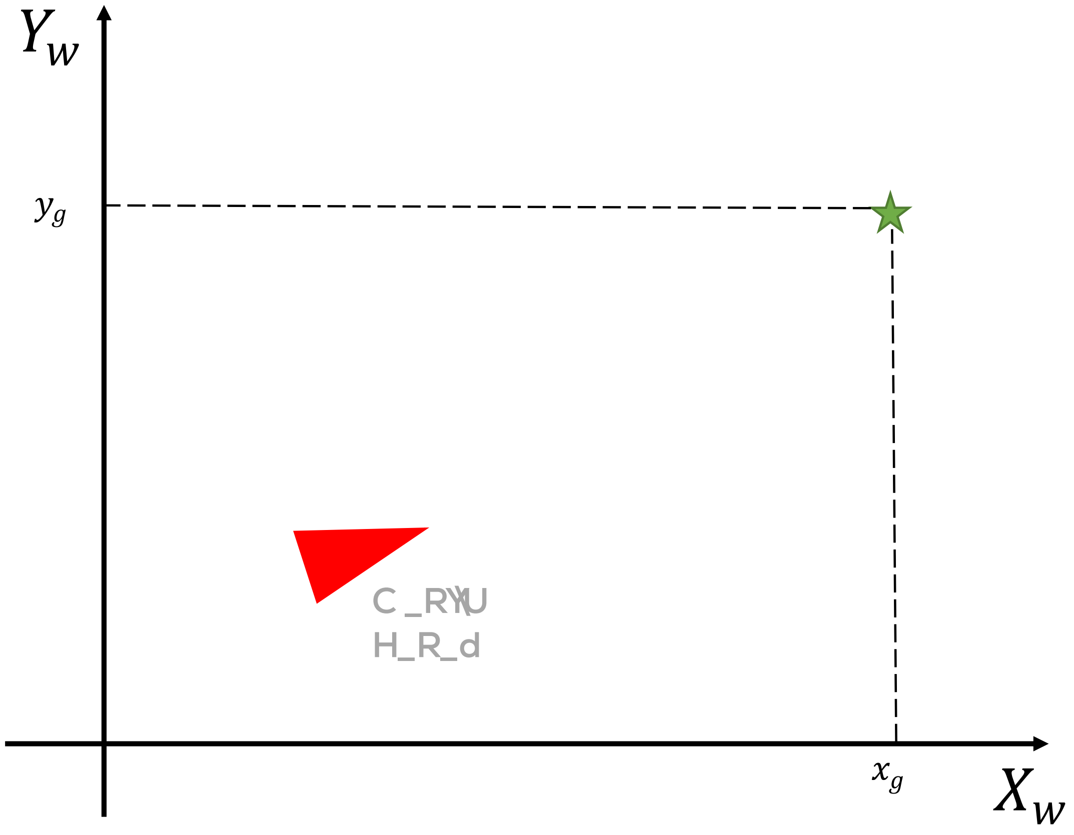
Point Stabilisation

Control Inputs

- The inputs to the robot are the linear and angular velocity of the robot

$$\mathbf{u} = \begin{pmatrix} V \\ \omega \end{pmatrix}.$$

- The linear velocity of the robot, V, is always oriented in the direction of x axis of the robot reference frame because of the non-holonomic constraint.
- The inputs are transformed into wheel velocities by the motors.



Point Stabilisation

Error

- To Error is the difference between the set point and the measurement.

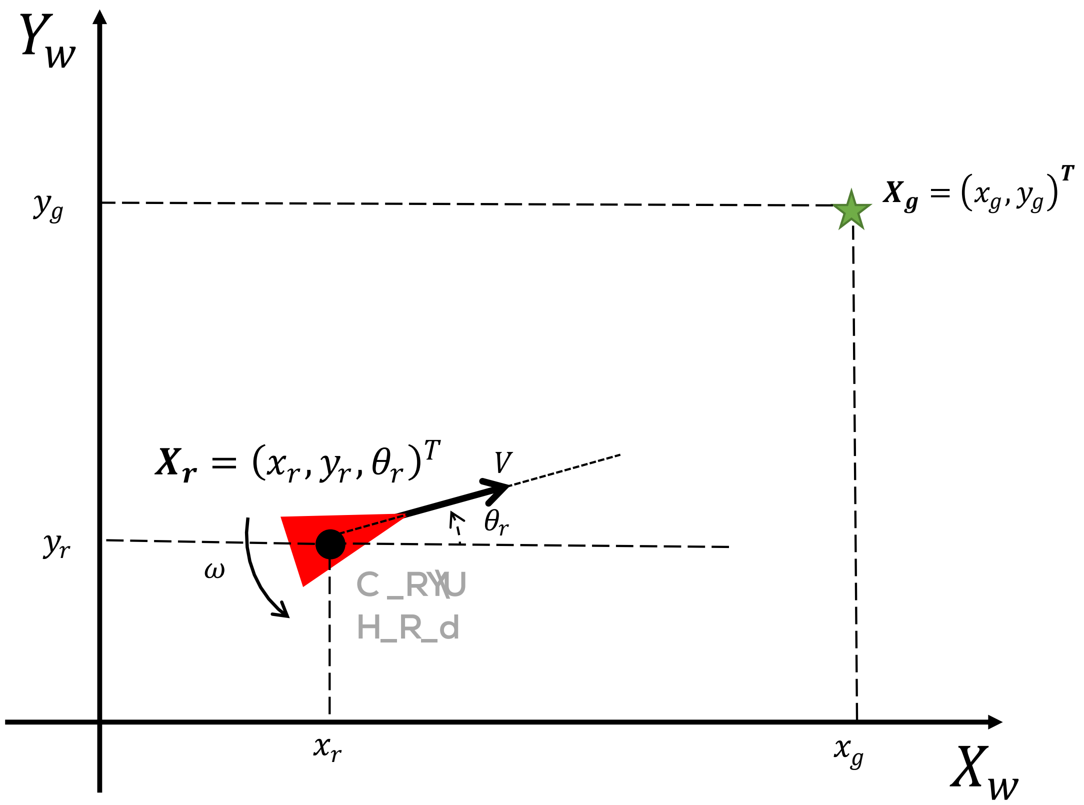
- In this case the set point was stablished to be $\boldsymbol{X}_g = \begin{pmatrix} x_g \\ y_g \\ \theta_g \end{pmatrix}$

- Therefore, for a non-holonomic robot moving in a 2D environment the error must be defined as:

$$\boldsymbol{e} = \boldsymbol{X}_g - \boldsymbol{X}_r$$

- Where, $\boldsymbol{X}_r$ represents the robot's pose represented by the vector

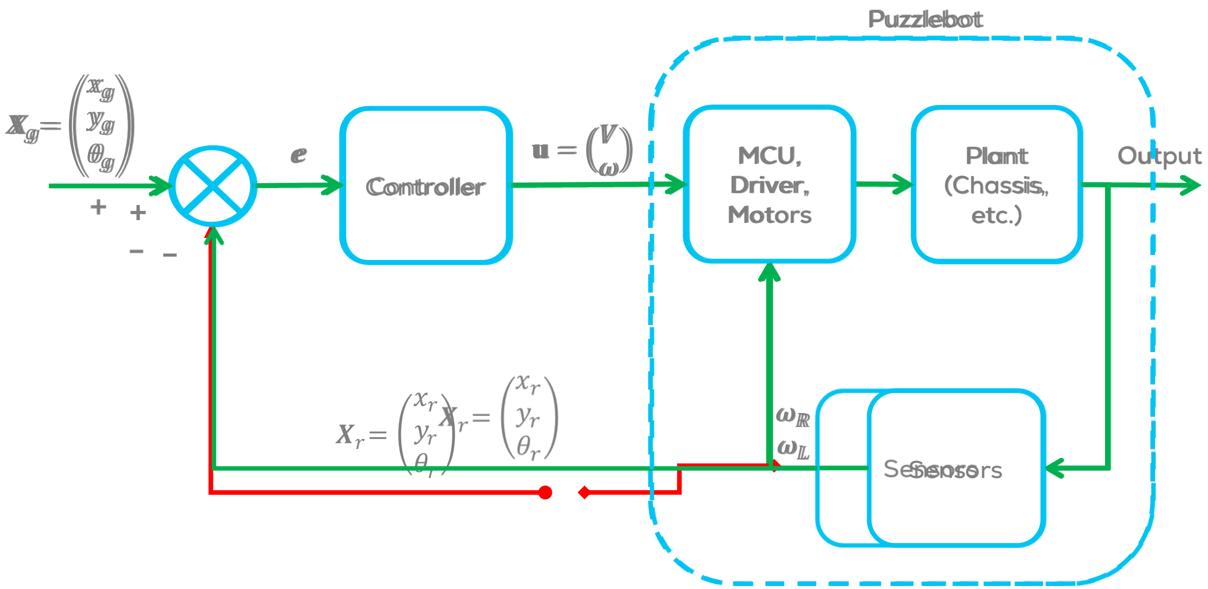
$$\boldsymbol{X}_r = \begin{pmatrix} x_r \\ y_r \\ \theta_r \end{pmatrix}.$$



The Control System

Control

- The control diagram can then be changed as follows…
- Just one problem…
- In reality, the Sensors of a Robot do not provide the position of the robot…
- Usually, the sensors of the mobile robot provide the information about the velocity of each wheel i.e., ω_R and ω_L.
- Another question arises… Can I use the information of the wheel's speed to get the position of my robot?
- YES… it's called Localisation!



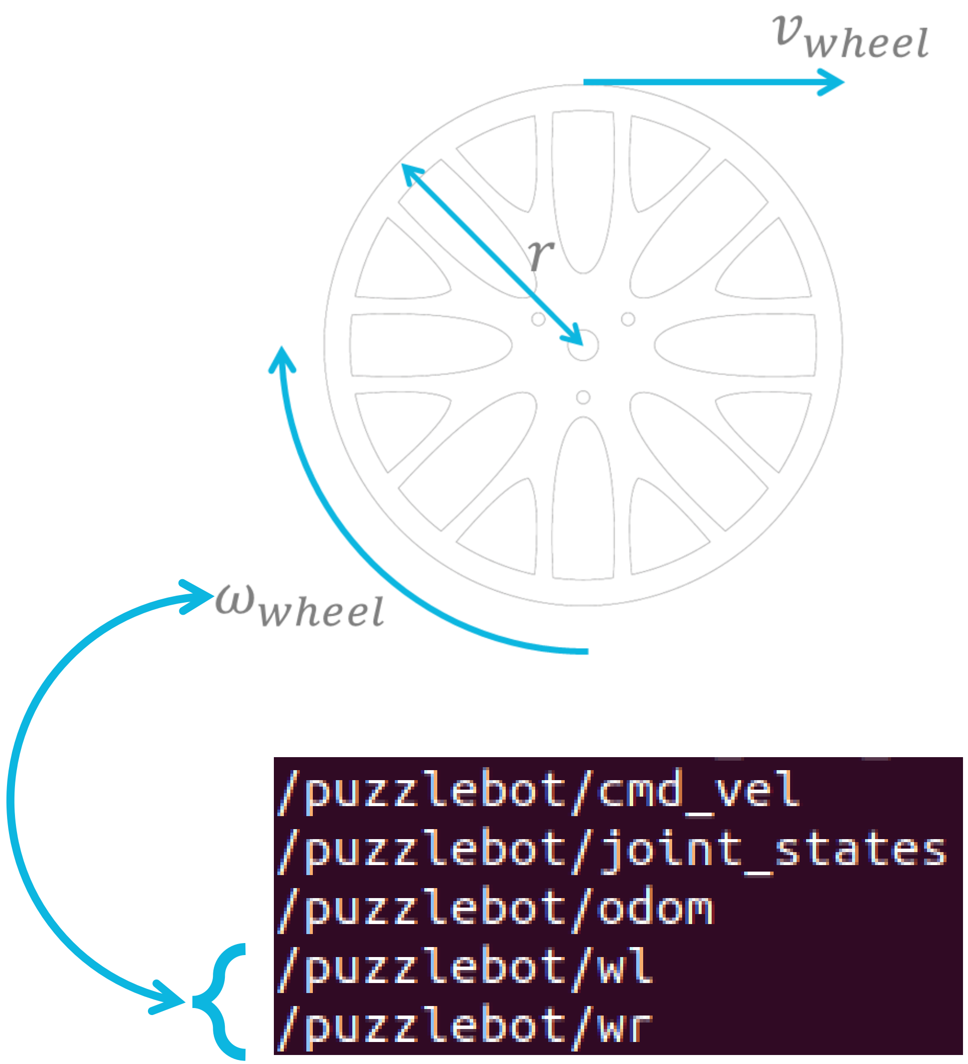
Localisation

Tangential Velocity

- The first step to get the position of a robot, is to get the tangential velocity of the wheel.
- The tangential speed of a wheel is given by

$$v_{wheel} = r \cdot \omega_{wheel}$$

- Where r is the radius and ω_{wheel} is the angular speed of the wheel.
- For this case, the robot has two wheels, therefore two tangential velocities must be calculated.
- On the Puzzlebot (and most robots) this is given by the topics `/wr` and `/wl`



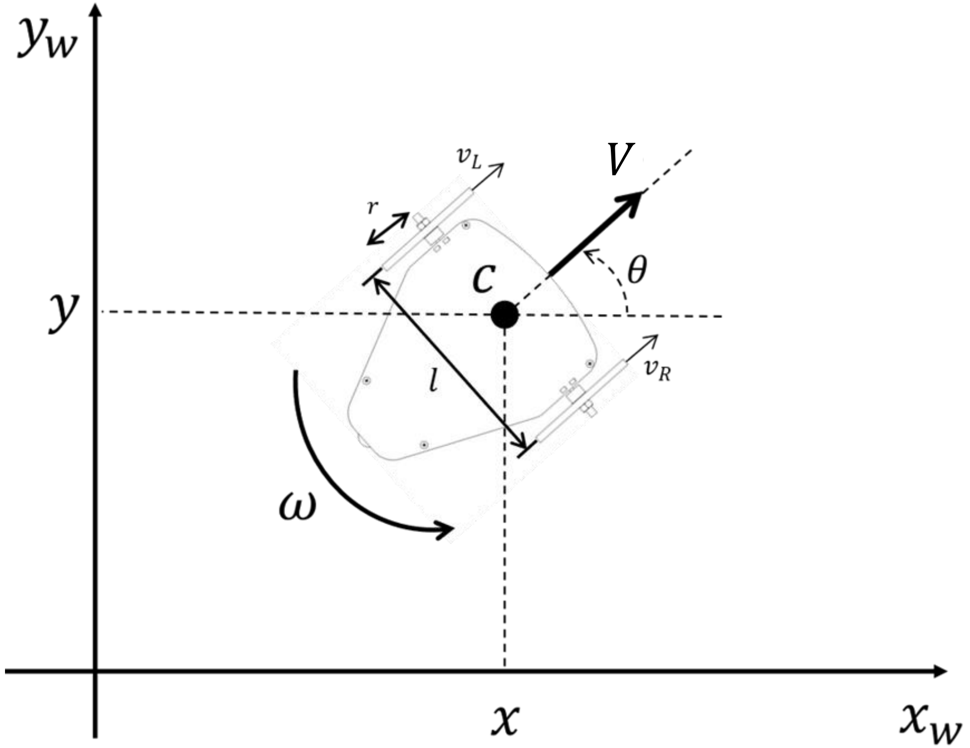

```
/puzzlebot/cmd_vel
/puzzlebot/joint_states
/puzzlebot/odom
/puzzlebot/wl
/puzzlebot/wr
```

Localisation

Robot Velocity

- Using the wheel velocities, it is possible to estimate the forward velocity V and the angular velocity ω.
- For this case, the resultant forward velocity V through C (centre of mass) may be reasoned as an average of the two forward wheel velocities given by

$$V = \frac{v_R + v_L}{2} = r\frac{\omega_R + \omega_L}{2}$$



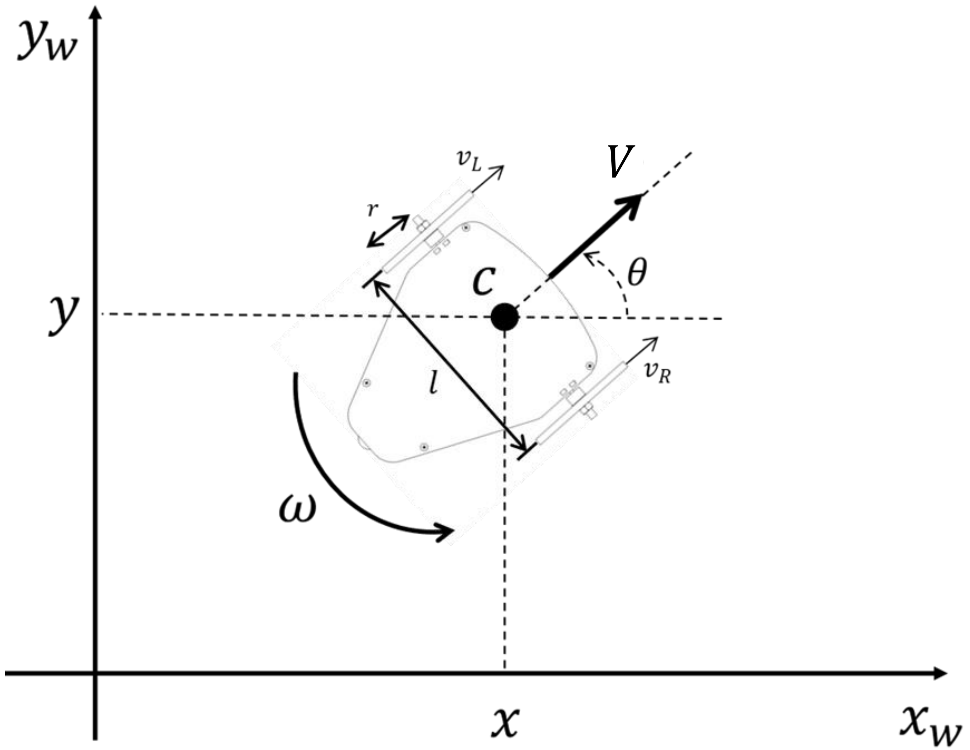
Differential Drive Robot
Puzzlebot ©.

Localisation

Robot Velocity

- 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.,

$$\omega = \frac{v_R - v_L}{l} = r\frac{\omega_R - \omega_L}{l}$$



Differential Drive Robot
Puzzlebot ©.

Localisation

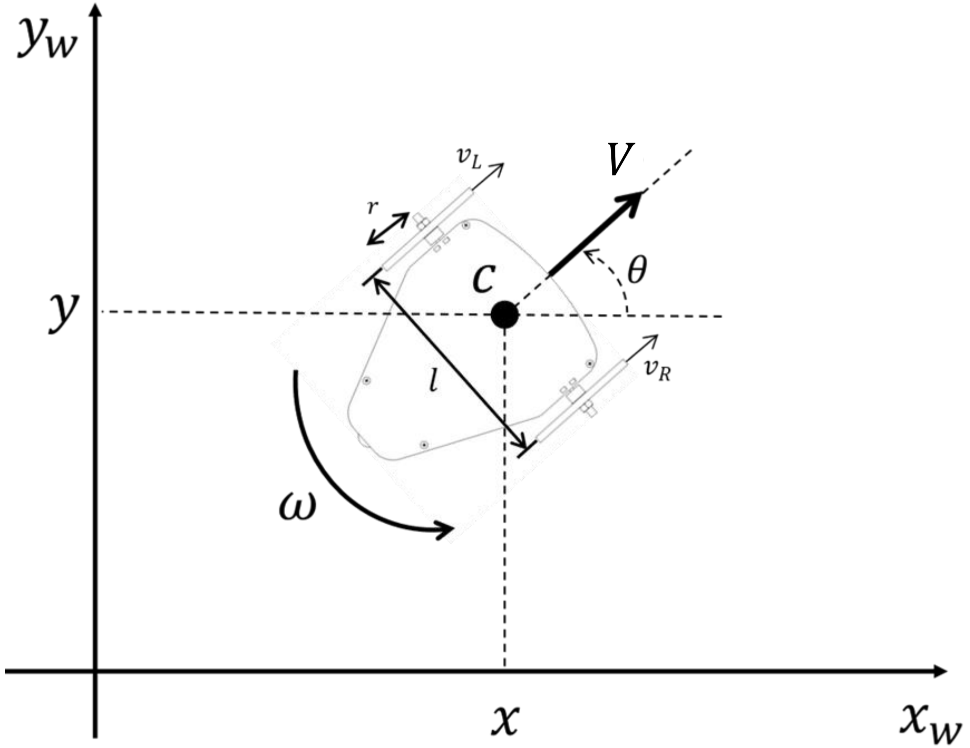
Robot Velocity

- Using the Differential Drive Kinematic Model, given by:

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

- It is possible to decompose the speed of the robot into its components

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



Differential Drive Robot
Puzzlebot ©.

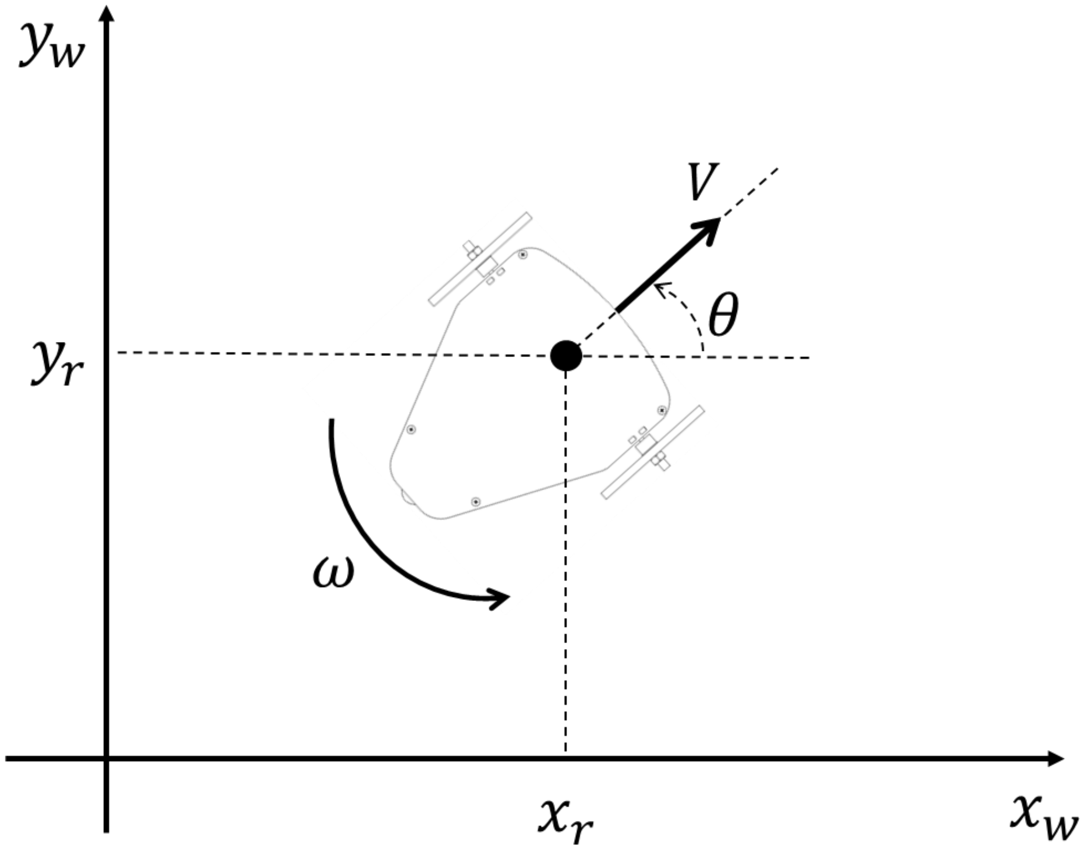
Localisation

Robot Position

- Discretising the model and solving it using Euler's Method:

$$\begin{cases} x_{r\,k+1} = x_{r\,k} + r\dfrac{\omega_R + \omega_L}{2}\ \cos\theta\ \ dt \\ y_{r\,k+1} = y_{r\,k} + r\dfrac{\omega_R + \omega_L}{2}\ \sin\theta\ \ dt \\ \theta_{r\,k+1} = \theta_{r\,k} + r\dfrac{\omega_R - \omega_L}{l}\,dt \end{cases}$$

- Where ω_R and ω_L are the speed given by the encoders
- Estimating the change in position a robot from its sensors is called "odometry".



Determining the Robot Position

$$\boxed{\theta_{r\,k+1}} = \boxed{\theta_{r\,k}} \boxed{+}\, r\frac{\boxed{\omega_R - \omega_L}}{\boxed{l}}\boxed{dt}$$

$$\boxed{x_{r\,k+1}} = \boxed{x_{r\,k}} \boxed{+}\, r\frac{\boxed{\omega_R + \omega_L}}{2}\boxed{dt}\cos\boxed{\theta_k}$$

$$\boxed{y_{r\,k+1}} = \boxed{y_{r\,k}} \boxed{+}\, r\frac{\boxed{\omega_R + \omega_L}}{2}\boxed{dt}\sin\boxed{\theta_k}$$

Robot Location:
(x_k, y_k, θ_k): Pose of the robot at timestep k (m, m, rad). Stored in memory, initial value 0

Robot Constants:
r: Wheel radius = 0.05 m
l: Distance between robot wheels = 0.19 m

Measured variables
$(\omega_{\mathrm{R}}, \omega_{\mathrm{L}})$: Wheel velocity (rad/s)
dt: Time between samples (s)

Values of θ can grow unbounded so they must be contained within a single circle (wrap2pi):
Either:

$$-\pi \le \theta < \pi$$

Or:

$$0 \le \theta < 2\pi$$

Point Stabilisation Control

- For the sake of simplicity, in this course, the goal is defined only by a 2D set of coordinates

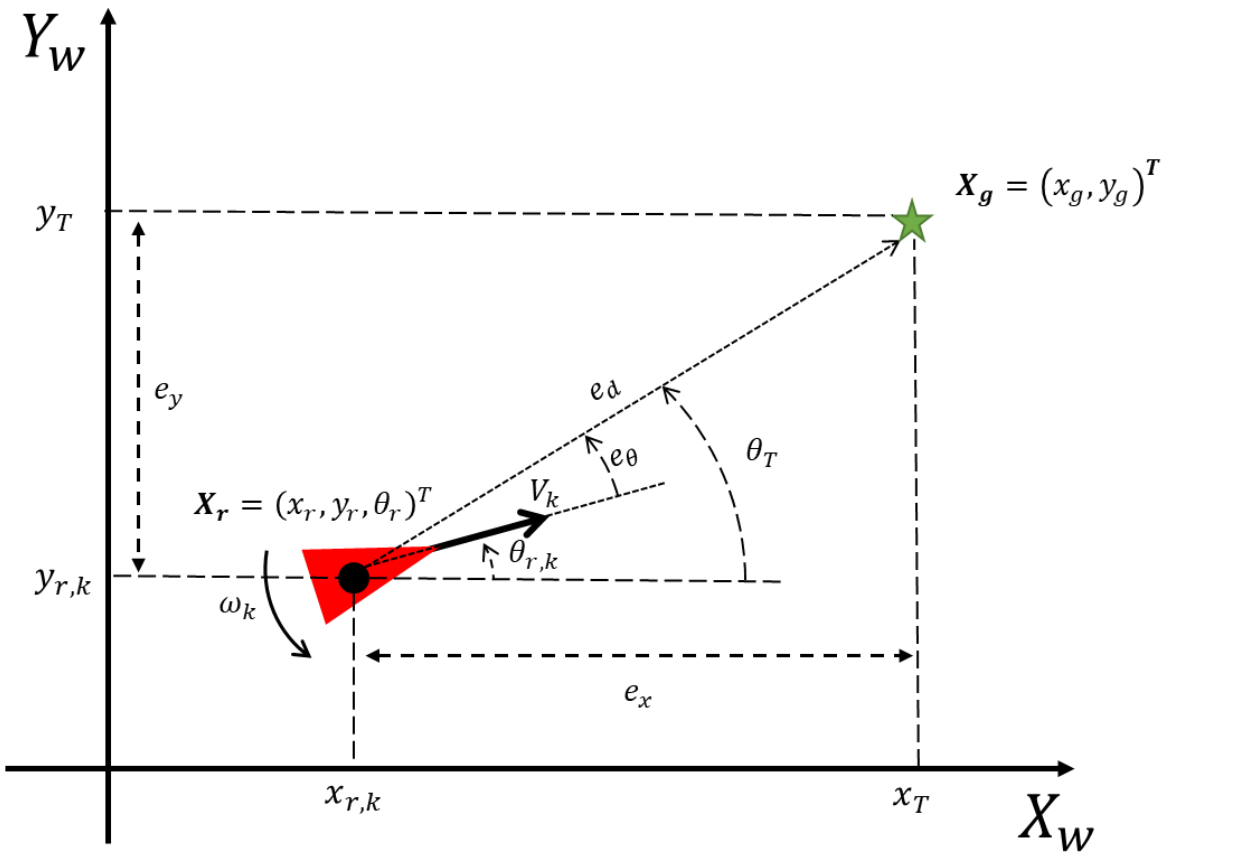
$$\boldsymbol{X}_g = \begin{pmatrix} x_g \\ y_g \end{pmatrix}.$$

- Then compute the errors by using the goal coordinates and robot position as in the following:

$$e_x = x_g - x_r$$

$$e_y = y_g - y_r$$

$$e_\theta = atan2(e_y, e_x) - \theta_r$$



Point Stabilisation

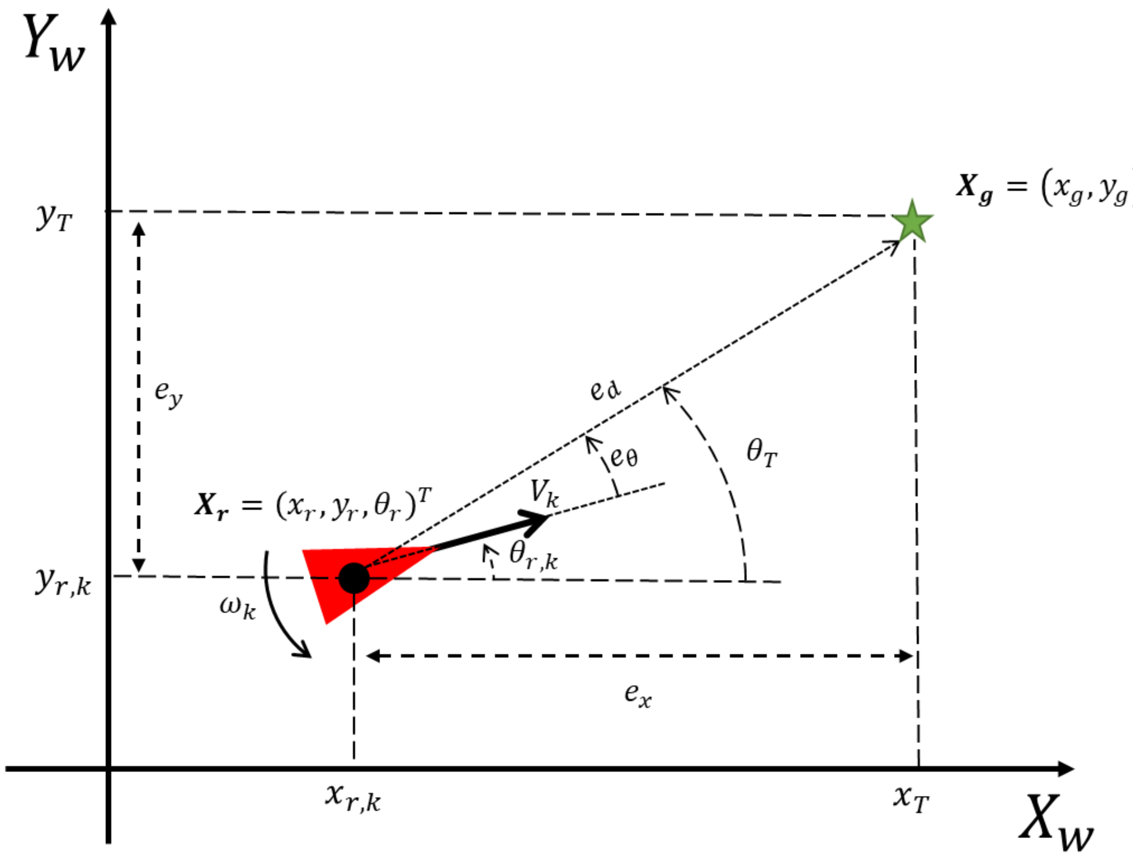
- For simplicity, the six controller gain parameters can be reduced to only two by defining the distance error:

$$e_d = \sqrt{e_x^2 + e_y^2}$$

- The control law can now be written as:

$$V = K_d e_d$$

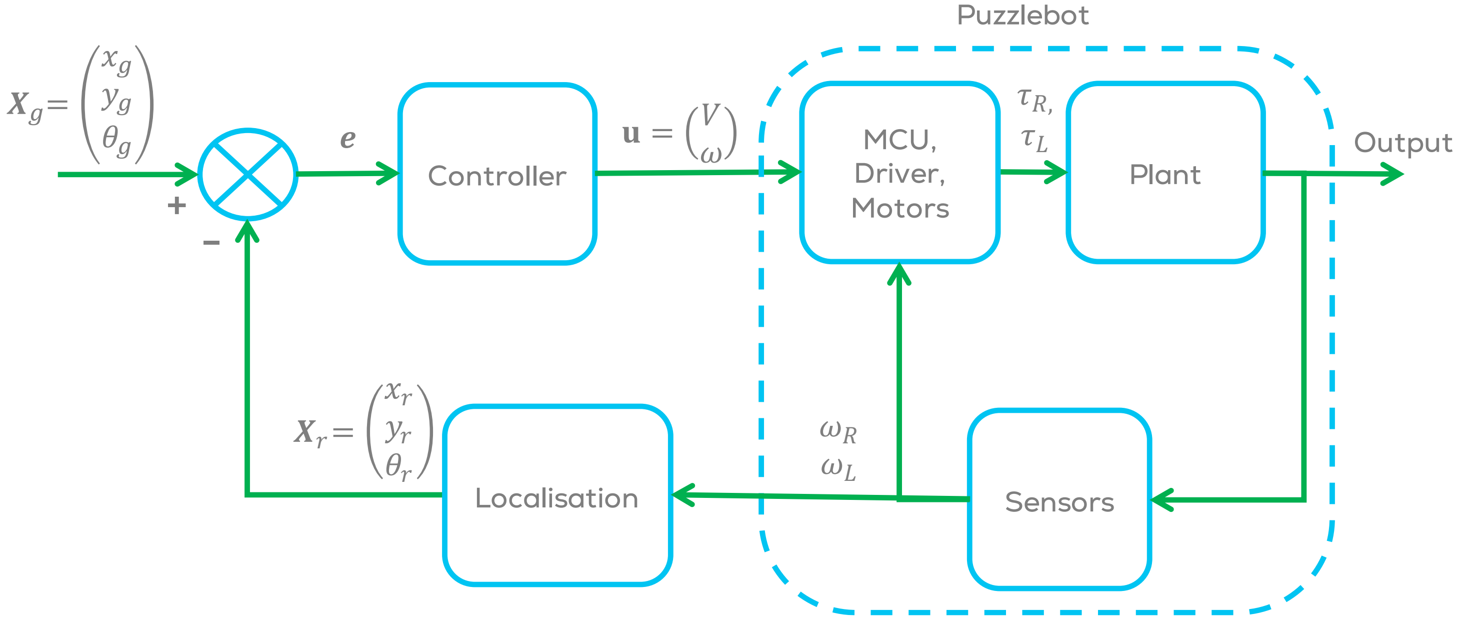
$$\omega = K_\theta e_\theta$$



The Control System

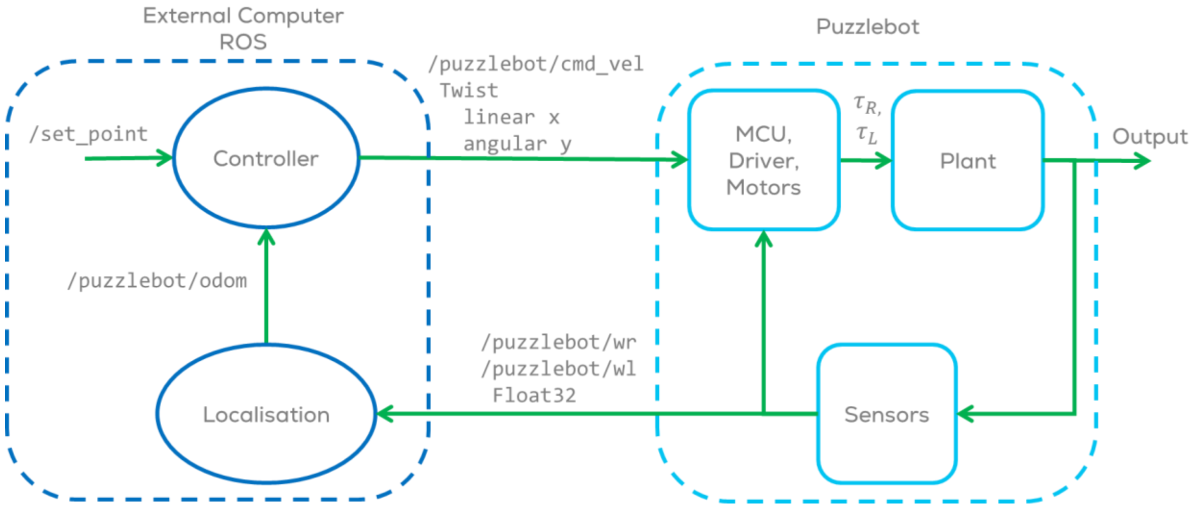
Control

- The control diagram can now be redefined as follows.
- It can be observed that the position can now be estimated using the encoder information.

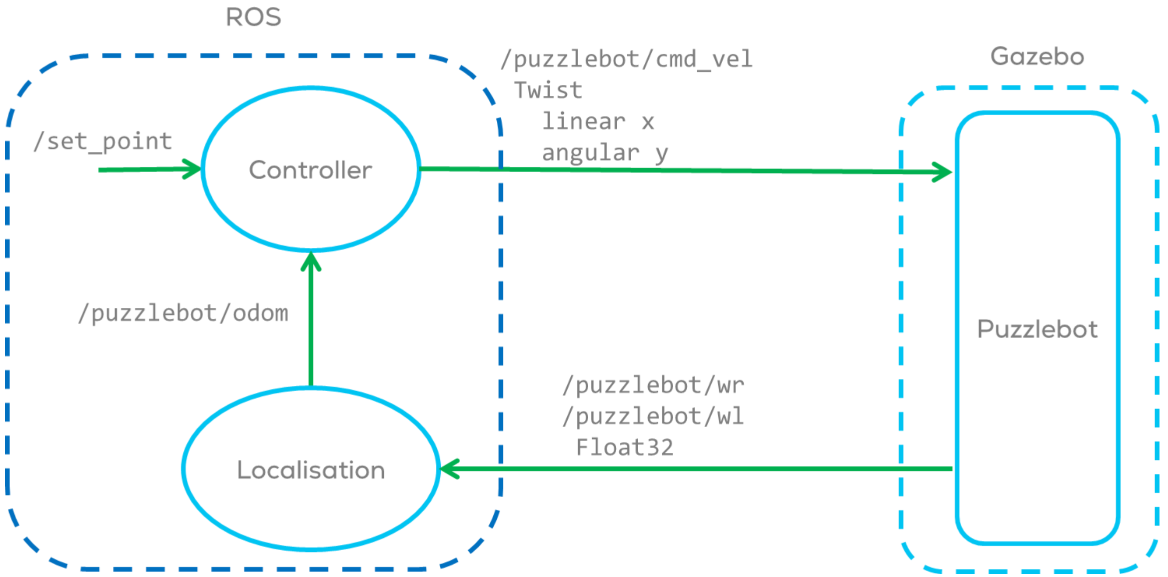


The Control System

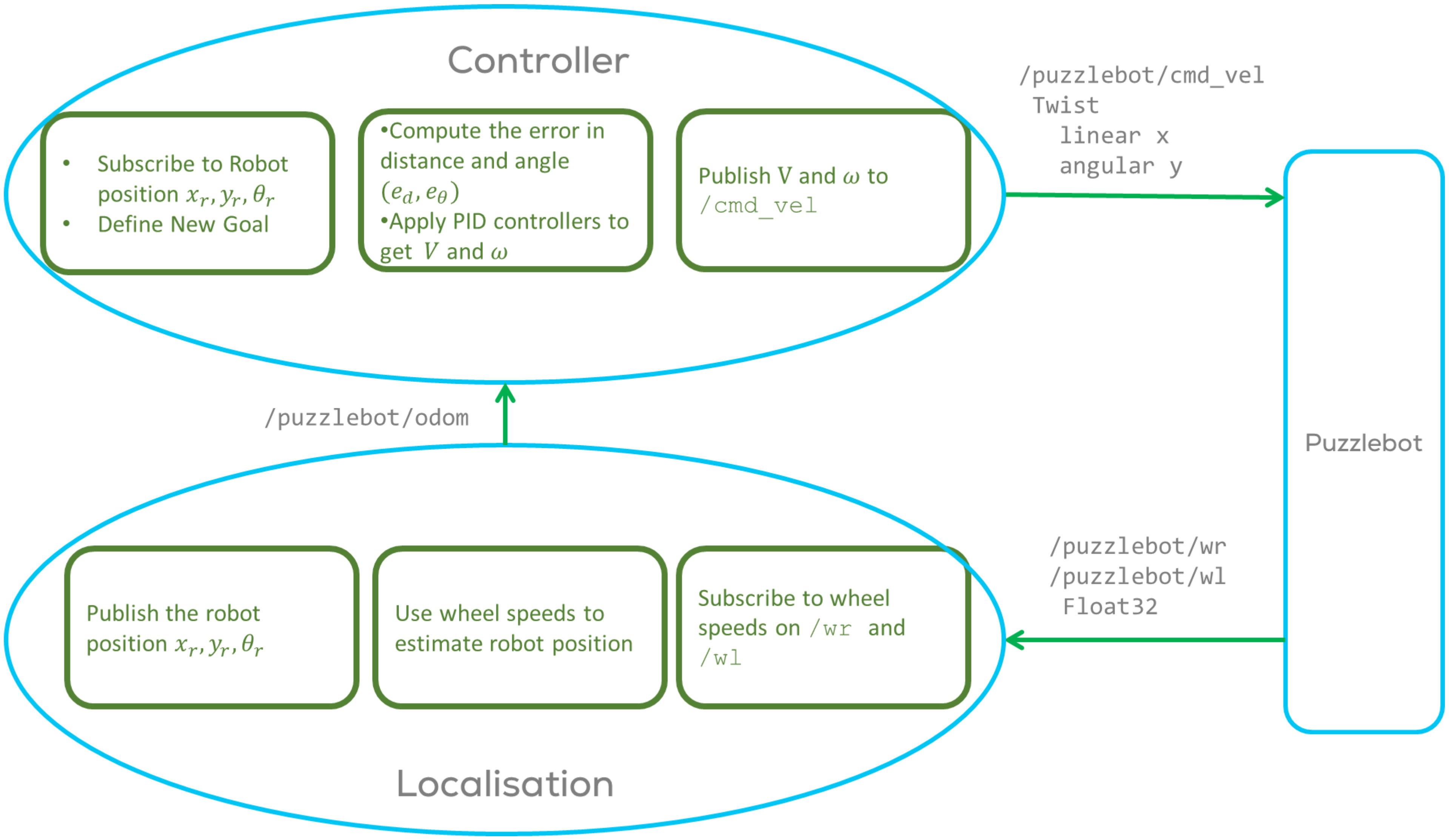
Real Robot



Gazebo sim



ROS setup


Controller
Subscribe to Robot position x_r, y_r, θ_r
Define New Goal
•Compute the error in distance and angle (e_d, e_θ)
•Apply PID controllers to get V and ω
Publish V and ω to /cmd_vel
/puzzlebot/cmd_vel
Twist
linear x
angular y
Puzzlebot
/puzzlebot/odom
/puzzlebot/wr
/puzzlebot/wl
Float32
Publish the robot position x_r, y_r, θ_r
Use wheel speeds to estimate robot position
Subscribe to wheel speeds on /wr and /wl
Localisation

Activity 1: Localisation Node

- Implement a ROS node that computes the robot location using the encoder data
  - It should subscribe to `/puzzlebot/wl` and `/puzzlebot/wr`, and publish the data to a suitable set of topics
  - The published messages could be a set floats, or you can use the Odom message (standard way).
  - Use the remote-control function “teleop_twist_keyboard” to test the position estimation.

Activity 2: The Controller

- Make another node to that computes e_d and e_θ.
- Use the remote-control function “teleop_twist_keyboard” to test the errors.
- Drive the robot around, checking that the angle to the target and the distance from the target are updated correctly
- Remember to wrap all angles (wrap to pi)

```
#wrap to pi function
def wrap_to_Pi(theta):
    result = np.fmod((theta + np.pi),(2 * np.pi))
    if(result < 0):
        result += 2 * np.pi
    return result - np.pi
```

Activity 2: The Controller

- Since the robot is inherently stable, a simple PID scheme should be sufficient.
- Start with a pair of proportional controllers:

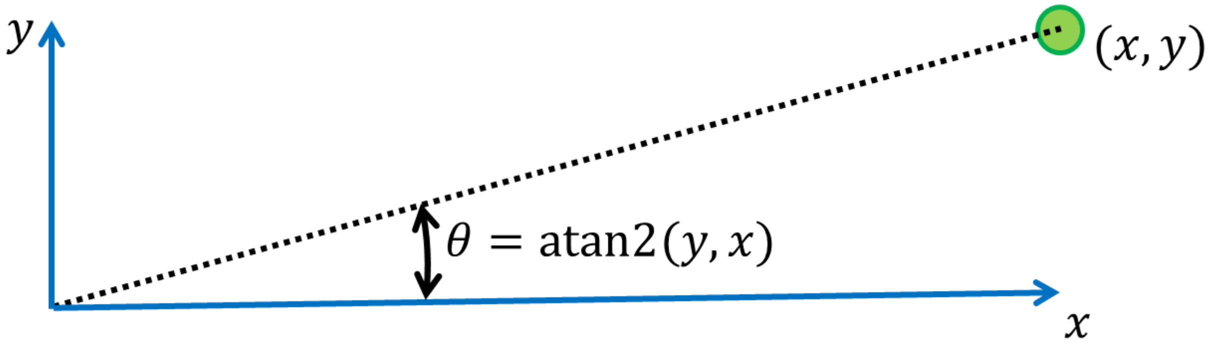
$$V = K_v e_d$$

$$\omega = K_\omega e_\theta$$

… and add integral and derivative elements if necessary?.

atan2

- The atan2 function is a special form of arctan or $\tan^{-1}$.
- It takes two arguments, y and x, and returns the angle to the x axis:



- It is included in most maths libraries, but it is recommended to use numpy, as numpy will be necessary later on in the course

```
import numpy
theta = numpy.arctan2(y,x)
```

Tips and Tricks

- Write and test your node with the Gazebo Simulator:
  - Use this to check the basics of your code are working correctly, such as the sign (+/-) of your controller parameters K_v and K_ω
  - Does the robot turn towards the goal?
  - Does the robot move towards or away from the goal?
- Tune one of the controllers at a time. You may find it easier to tune K_ω first, while setting your robot to move with a fixed forward speed.
- If in doubt, *lower* the value of the control constants.
- You may find it helpful to use a launch file to load your controller constants using parameters.

Accuracy

- It will not be possible to tune the controllers such that the robot moves perfectly into position.
  - You will need a threshold after which your algorithm decides it has successfully arrived.
  - Suggested initial threshold: 10 cm ($e_d < 0.1\ m$)
- Additionally, if you measure the position of the robot, it will likely not match up with the measurement computed from the encoders.
  - This is inevitable due to additive noise in the encoder readings.
  - The solution to this is to use sensors that can measure the position of the robot relative to its environment (another class).