Locomotion of a Mobile Robot in a 2D Space
Ackermann Steering

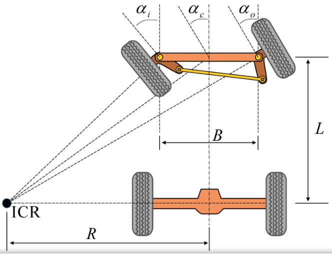
Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory

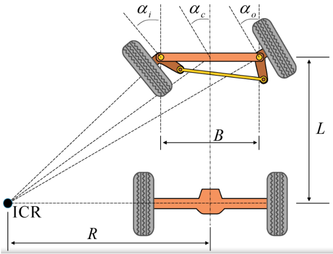
Dipartimento di Matematica e Informatica - Università di Catania, Italy

santoro@dmi.unict.it

Robotic Systems



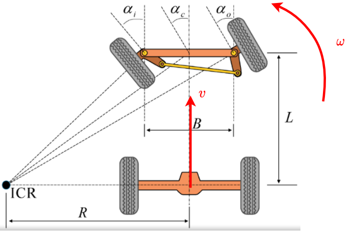
- It is the locomotion model of the cars
- It is based on a (rear or front) single traction actuator (motor) plus a mechanism to steer front wheels
- The traction (rear in figure) is connected to a gearbox called **differential** that allow traction wheels to rotate at different speeds during turns



Geometric Model

- *ICR*: Instantaneous Center of Rotation
- *R*: Radius of Rotation
- *B*: Wheelbase
- *L*: Lateral Wheelbase
- α_i: Inner wheel steering angle
- α_o: Outer wheel steering angle
- α_c: Center (virtual) wheel steering angle

Ackermann Steering

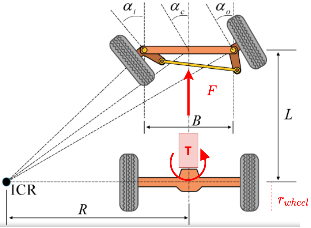


Kinematics

- In Ackermann steering vehicle the speeds of the rigid body (v, ω) **are not independent**
- ω depends on v and the steering angle α_c

$$\omega = \frac{v}{R} \qquad R = \frac{L}{\tan \alpha_c}$$

Ackermann Steering



Dynamics

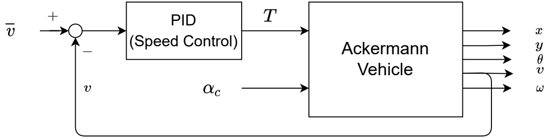
- In Ackermann steering vehicle the motor generates a **torque** T that, on the basis of the radius of traction wheels, becomes a traction **force**
- The dynamics can be modeled in a similar way as to what we did for the cart

$$F = \frac{T}{r_{wheel}} \qquad \dot{v} = -\frac{b}{M} v + \frac{1}{M} F$$

Ackermann Steering

lib/models/cart2d.py

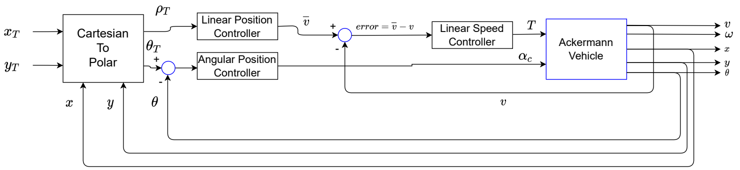
```
class AckermannSteering:

    def __init__(self, _mass, _lin_friction,
                       _r_traction, _lateral_wheelbase):
        self.M = _mass
        self.b = _lin_friction
        self.r_wheels = _r_traction
        self.l_wb = _lateral_wheelbase
        self.v = 0
        self.w = 0
        self.x = 0
        self.y = 0
        self.theta = 0

    def evaluate(self, delta_t, torque, steering_angle):
        _force = torque / self.r_wheels
        new_v = self.v * (1 - self.b * delta_t / self.M) + \
                    delta_t * _force / self.M
        if steering_angle == 0:
            new_w = 0
        else:
            curvature_radius = self.l_wb / math.tan(steering_angle)
            new_w = new_v / curvature_radius

        self.x = self.x + self.v * delta_t * math.cos(self.theta)
        self.y = self.y + self.v * delta_t * math.sin(self.theta)
        self.theta = self.theta + delta_t * self.w
        self.v = new_v
        self.w = new_w
```



Speed Control

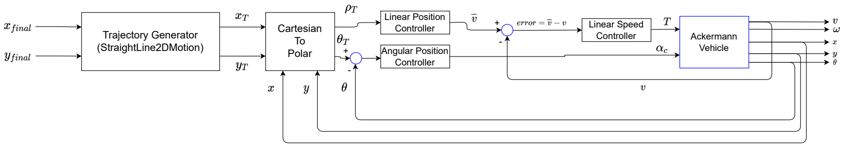
- The linear speed v can be directly controlled using a classical PID
- Angular speed ω **cannot be** directly controlled since it depends on both the steering angle and the linear speed

Ackermann Steering



Position Control

- The polar position control can be directly used by considering steering angle instead of angular speed
- The output of the angular position controller is not the target ω but the **steering angle** α_c
- Remember that that both linear and angular position controllers are P-controllers with saturation



Trajectory Following

- The polar position control can be in turn driven by the **trajectory generator** in order to follow a certain straight line towards a final point

Locomotion of a Mobile Robot in a 2D Space
Ackermann Steering

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it

Robotic Systems