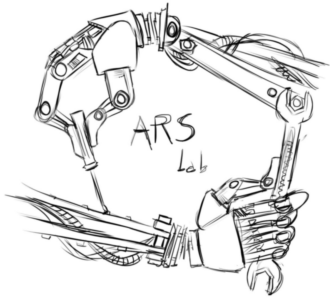
Modellazione e simulazione di un robot mobile su ruote

Corrado Santoro

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



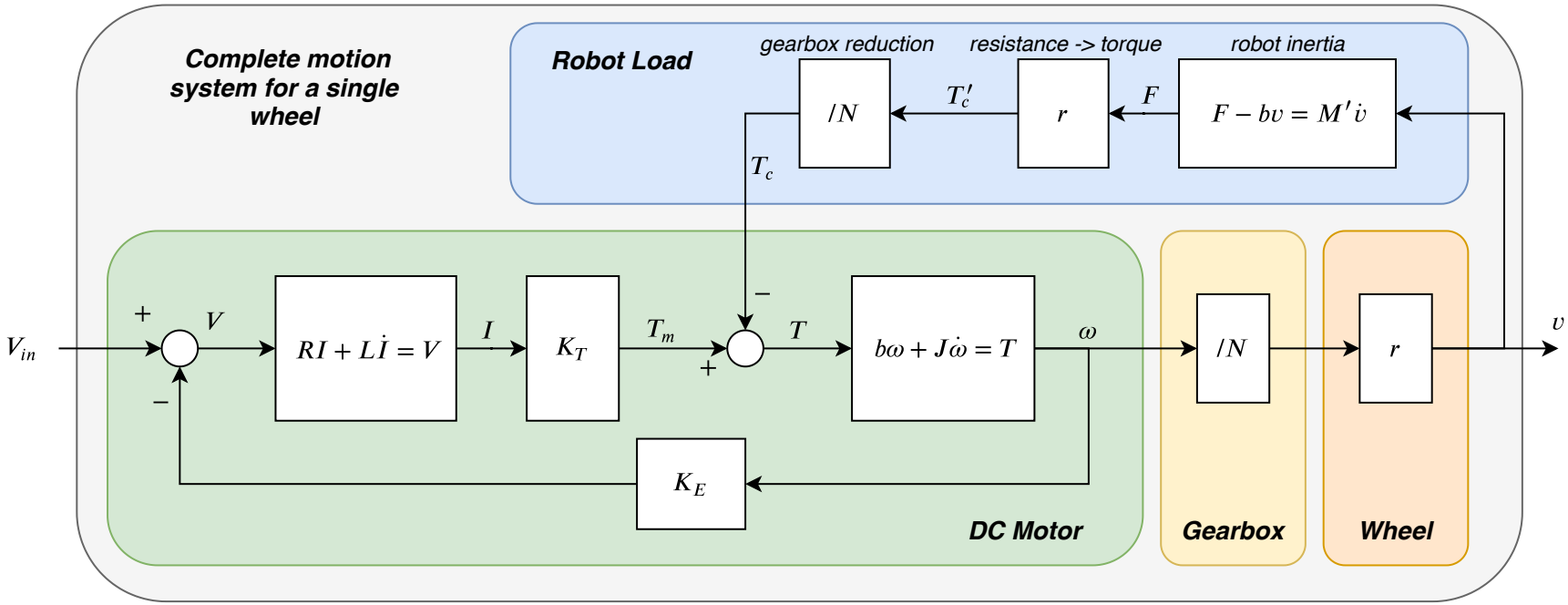
Programmazione Sistemi Robotici

Step di Modellazione Robot Mobile su Due Ruote

1. Modellazione del singolo **motore di trazione**, considerando il carico (peso robot) che insiste su esso
2. Modellazione del **sistema di locomozione**, composto da due motori più i relativi controlli di velocità delle singole ruote
3. Modellazione e implementazione dei vari **algoritmi di controllo del moto**

Modellazione del singolo motore di trazione

Modello Motore e Carico Associato

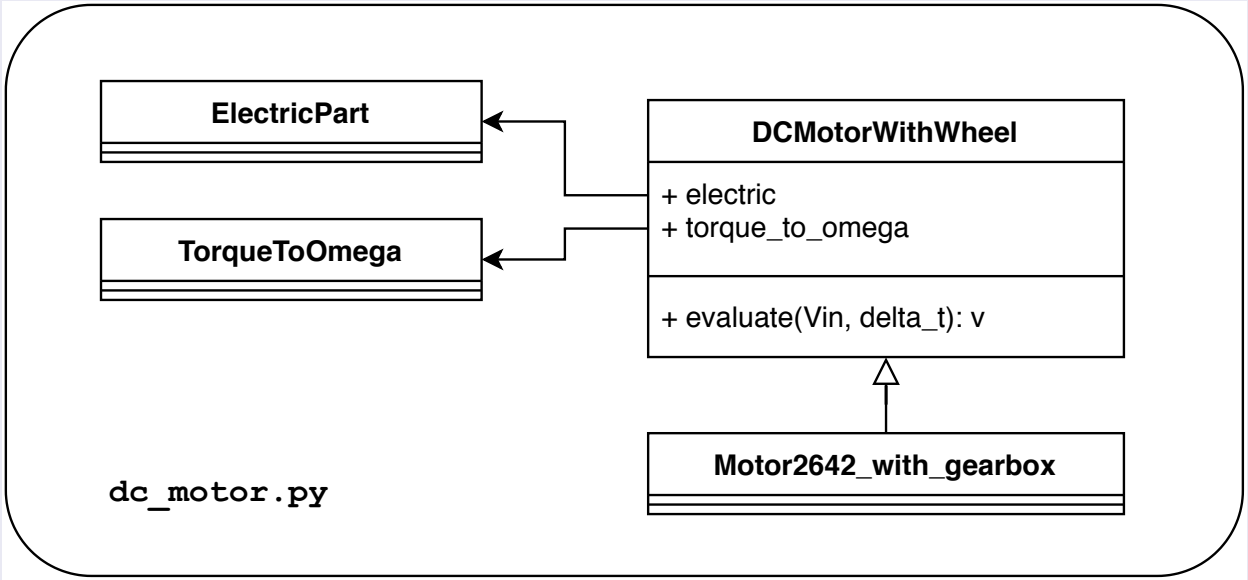


Il modello considera la presenza di:

- Riduttore di giri (**gearbox**), con $N =$ fattore di riduzione
- Ruota di trazione, **conversione** $\omega \rightarrow v$, con $r =$ raggio della ruota
- Carico indotto dal peso del robot, dove $M' = \frac{M}{2}$, con M massa del robot

Modello Motore e Carico Associato: Implementazione

File: `dc_motor.py`, Diagramma UML



Modello Motore e Carico Associato: Implementazione

File: dc_motor.py

```
# parte elettrica
class ElectricPart:  ...

# blocco coppia -> velocita' angolare
class TorqueToOmega: ...

# motore completo con carico
class DCMotorWithWheel:
    # _R, resistance of the inductor
    # _L, impedence of the inductor
    # _Kt, torque coefficient
    # _Ke, BackEMF coefficient
    # _gear, Gearbox ratio
    # _Wm, Wheel Mass
    # _Wr, Wheel Radius
    # _M, robot mass part
    # _b, Friction coefficient
    def __init__(self, _R, _L, _Kt, _Ke, _gear, _Wm, _Wr, _M, _b):
        ....

    def evaluate(self, Vin, delta_t):
        ...
        return v
```

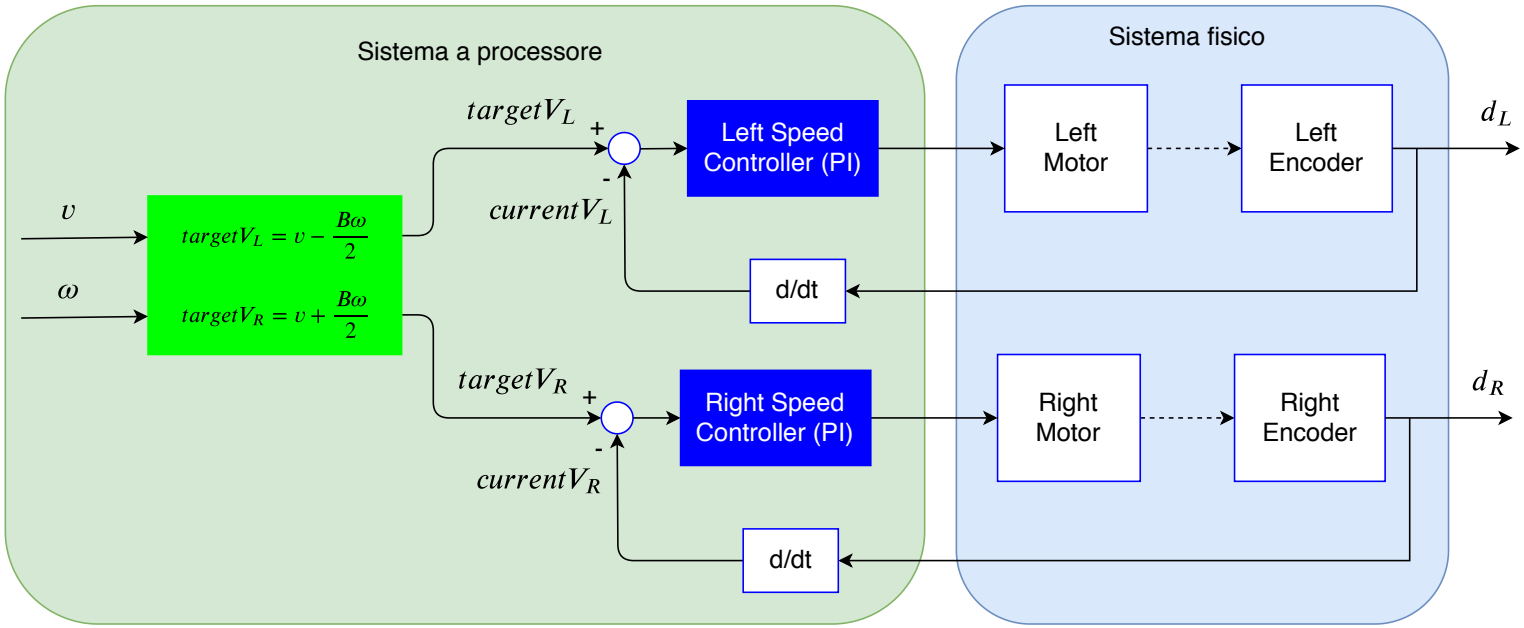
Modello Motore e Carico Associato: Implementazione

File: dc_motor.py

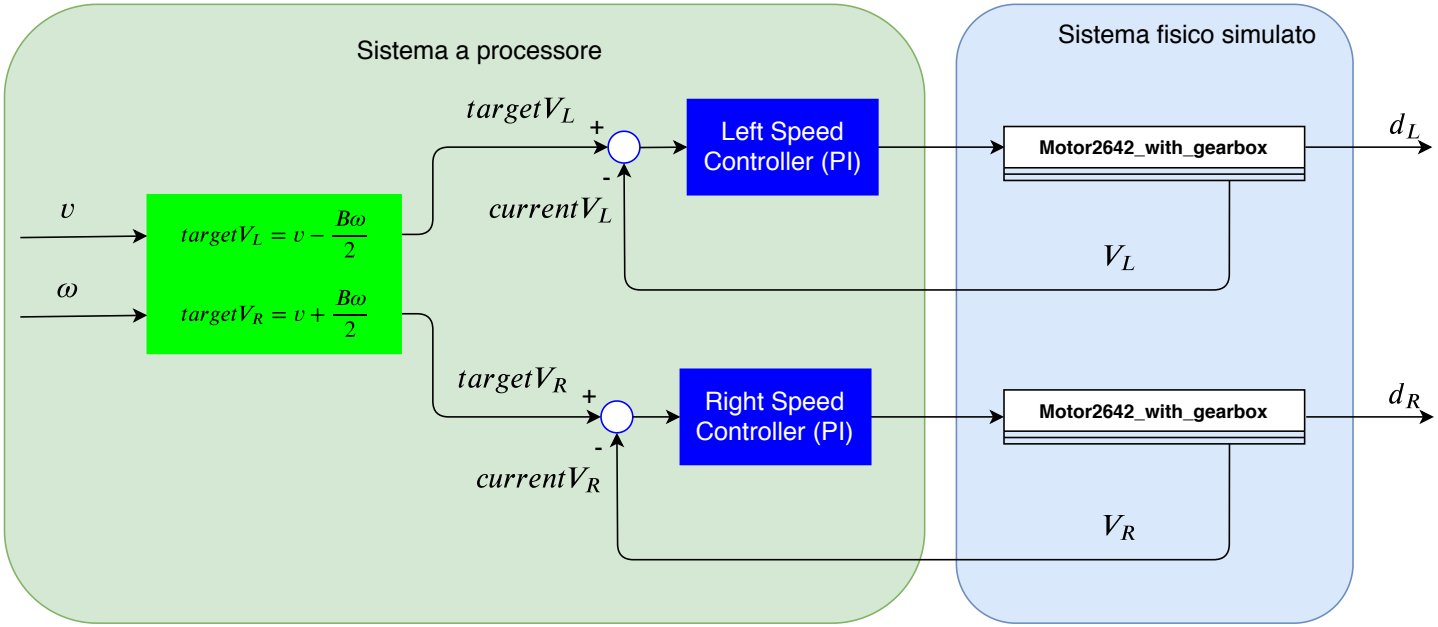
```
    ...
#
# motore Faulhaber 2642 con riduttore 14:1
#
class Motor2642_with_gearbox(DCMotorWithWheel):
    # _Wm, Wheel Mass
    # _Wr, Wheel Radius
    # _M, robot mass part
    # _b, Friction coefficient
    def __init__(self, _Wm, _Wr, _M, _b):
        DCMotorWithWheel.__init__(  self,
                                    1.45,      # R, 1.45 ohm
                                    130e-6,    # L, 130 microHenry
                                    0.0169,    # Kt, torque constant 16.9 mNm/A
                                    (60.0 * 0.00177)/(2*math.pi),
                                               # Ke, back EMF constant, 1.77 mV/rpm
                                    14.0,      # gearbox 14:1
                                    _Wm,       # Wheel Mass
                                    _Wr,       # Wheel Radius, 24mm
                                    _M,        # robot mass part
                                    _b )       # Friction coefficient
```

Modellazione del sistema di locomozione

Modello del sistema di locomozione

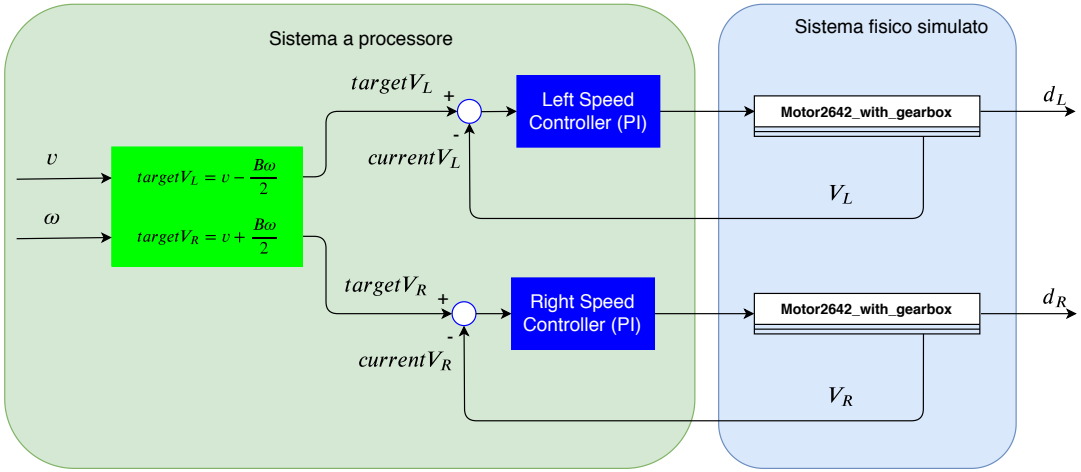


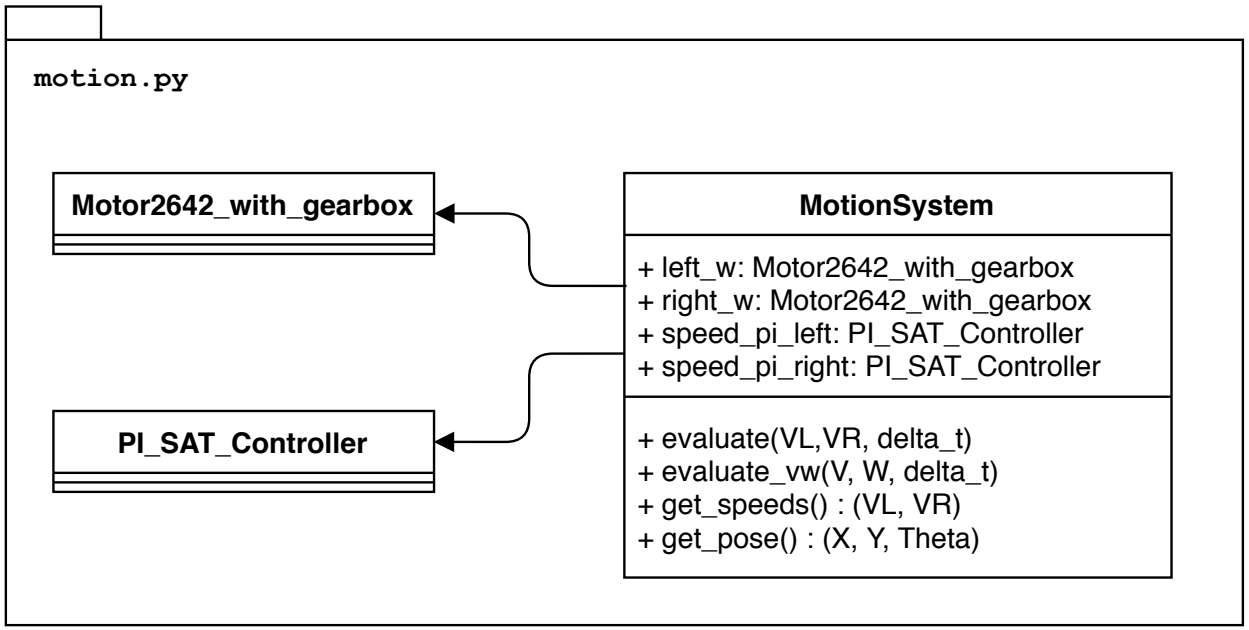
Modello del sistema di locomozione



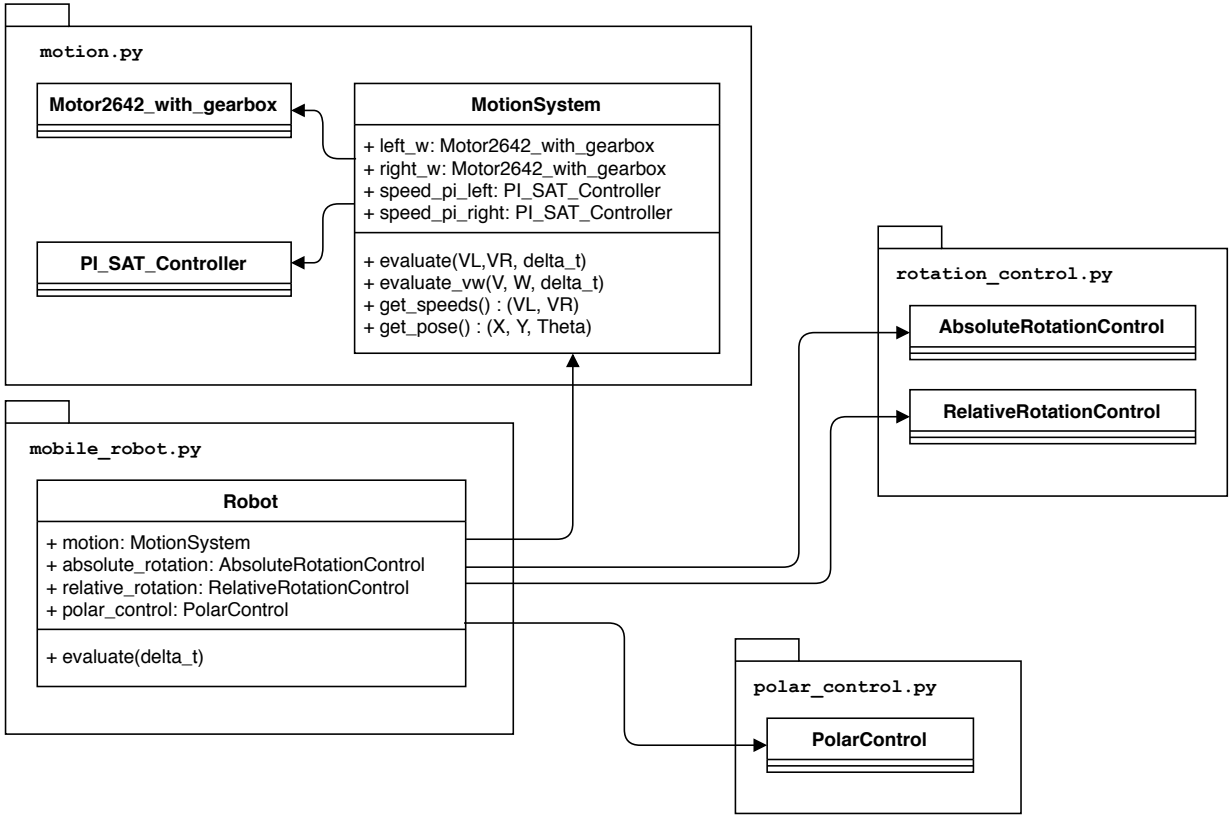
- Sostituiamo la parte del sistema fisico con due istanze delle classi che simulano le due ruote di locomozione
- Implementiamo i due controlli di velocità
- Incapsuliamo il tutto all'interno di una classe **`MotionSystem`**

Modello del sistema di locomozione



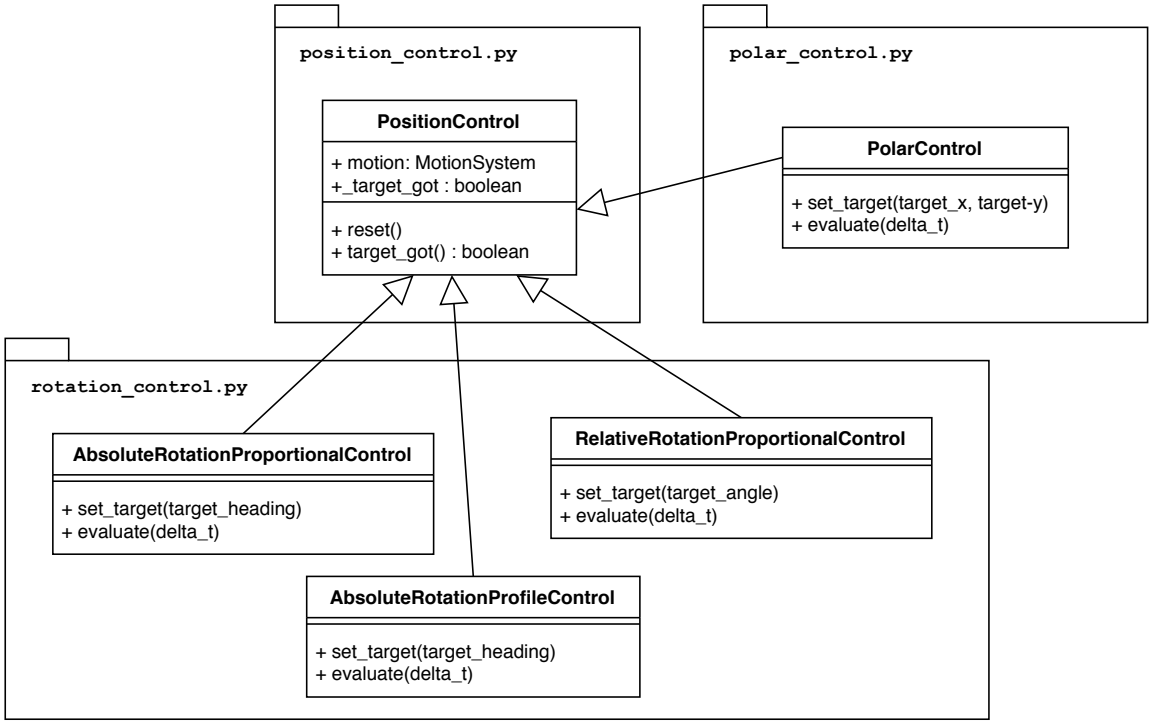


Inclusione del controllo in posizione



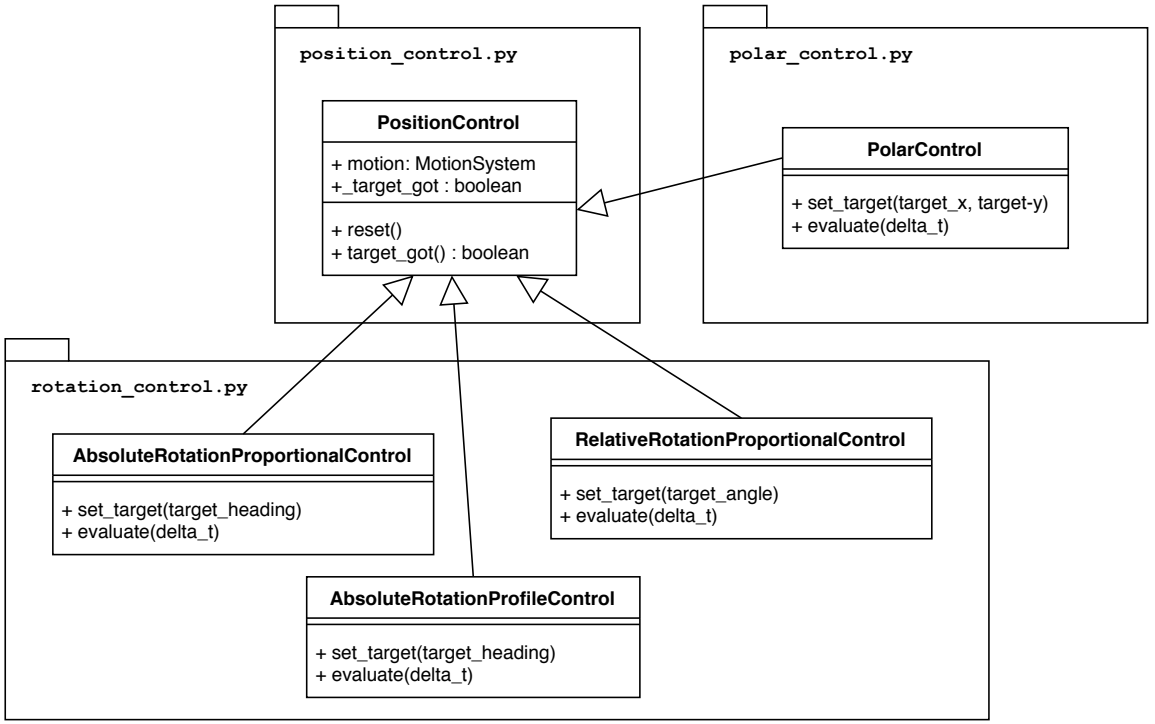
- Implementiamo i vari controlli in posizione, ognuno in una classe specifica
- Costruitamo una classe **`Robot`** che incapsula sia la parte di locomozione, sia i vari possibili algoritmi di controllo in posizione

Controlli in posizione specifici



- Costruiamo una super-classe **`PositionControl`** che verrà ereditata da tutte le classi che implementano gli specifici controlli in posizione
- Tale classe contiene il riferimento all'oggetto di tipo **`MotionSystem`** che rappresenta il sistema di locomozione
- Tale classe possiede anche l'attributo booleano **`_target_got`** che serve a rappresentare la condizione di target raggiunto

Controlli in posizione specifici

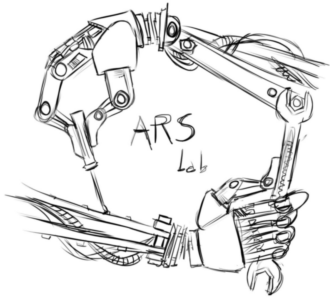


- Ogni specifico controllo esporrà i metodi:
  - **`set_target`**, per impostare il target specifico
  - **`evaluate`**, per effettuare uno step di controllo
- Quest'ultimo metodo dovrà anche occuparsi di impostare correttamente l'attributo **`_target_got`**

Modellazione e simulazione di un robot mobile su ruote

Corrado Santoro

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



Programmazione Sistemi Robotici