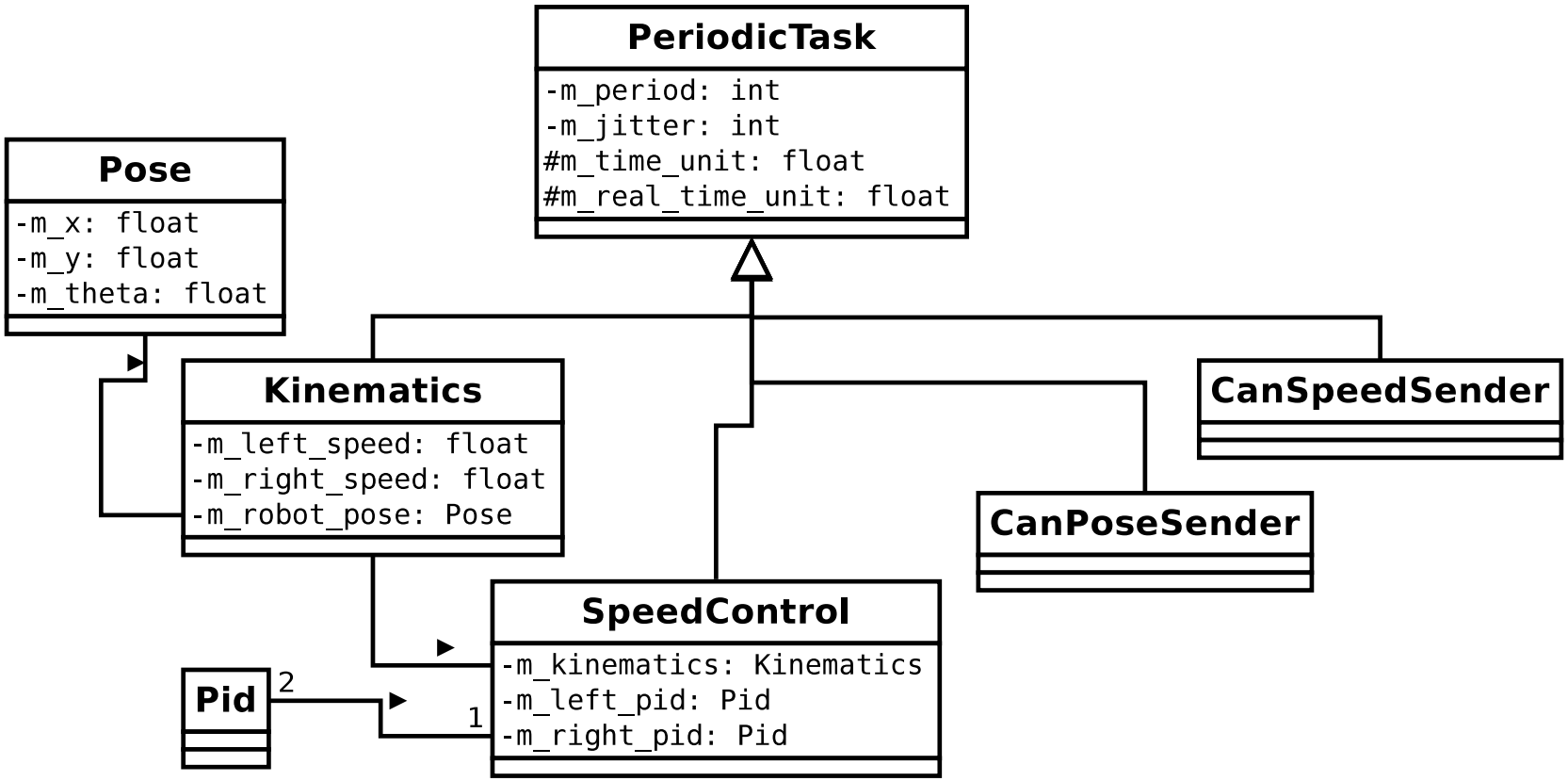
Architettura Software del Motion Control

Corrado Santoro

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

Programmazione Sistemi Robotici

Class Diagram



- Il software del motion control e basato su un insieme di **task periodici** ognuno dei quali effettua le attivita di gestione e controllo della movimentazione.
- In più sono presenti alcune classi che rappresentano informazioni aggiuntive sullo stato del robot

Classe Pose

La classe **Pose**, definita nel file `geometry.h`, rappresenta le informazioni sulla **posizione** del robot. Essa è derivata dalla classe più generica **Point**.

```
class Point {
 public:
   Point(float x = 0, float y = 0);
    ~Point() {};
    float x() { return m_x; };
    float y() { return m_y; };
    void x(float _x) { m_x = _x; };
    void y(float _y) { m_y = _y; };
    void operator+=(Point & p) { m_x += p.m_x; m_y += p.m_y;};
 private:
    float m_x, m_y;
};

class Pose : public Point {
 public:
 Pose(float x = 0, float y = 0, float theta = 0);
    float theta() { return m_theta;};
    void theta(float t) { m_theta = t; };
    void local_to_global(Point & robot_local, Point & global_point);
    void global_to_local(Point & global_point, Point & robot_local);
 private:
    float m_theta;
};
```

Classe Pose. Metodi.

- `float x();`
- `float y();`
- `float theta();`
  - Restituisce la rispettiva coordinata

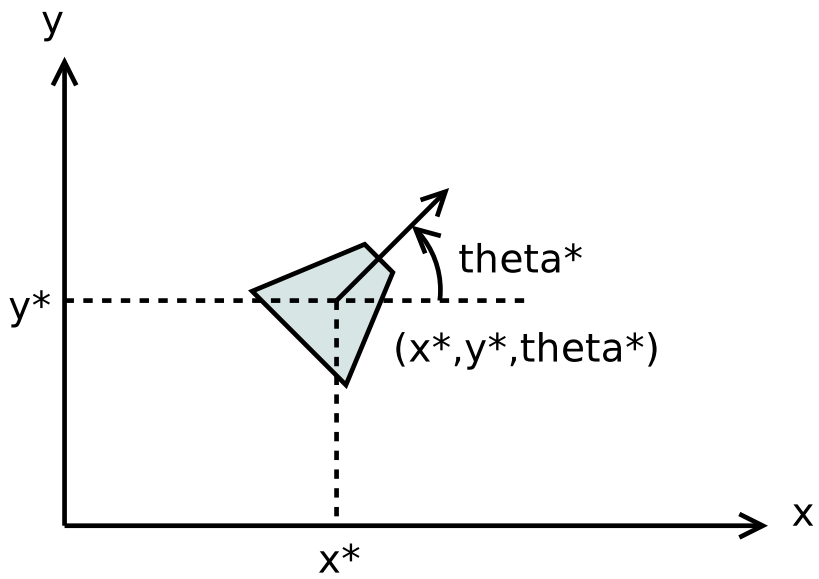
- `void x(float new_x);`
- `void y(float new_y);`
- `void theta(float new_theta);`
  - Imposta la rispettiva coordinata

- `void operator+=(Point & p);`
  - Somma tra punti

Classe Pose. Metodi.

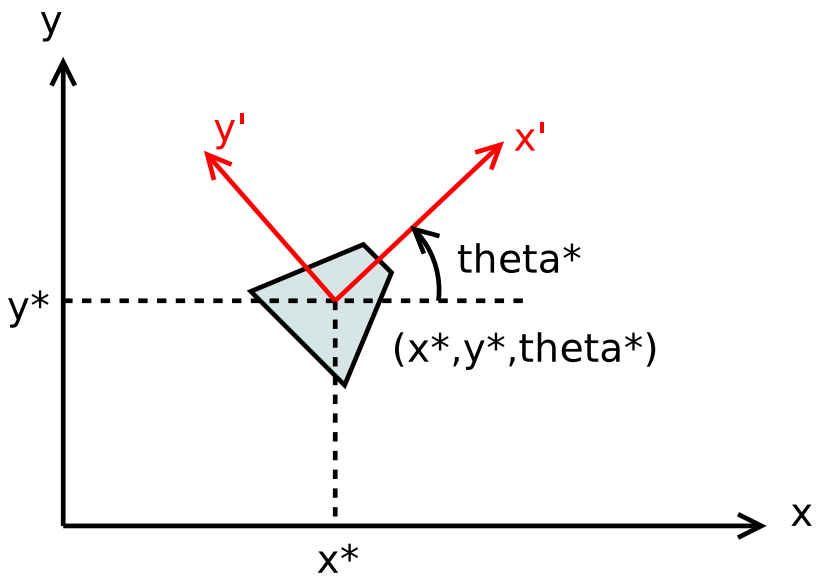
- `void local_to_global(Point & robot_local, Point & global_point);`
  - Converte un punto da coordinate locali del robot a globali
- `void global_to_local(Point & global_point, Point & robot_local);`
  - Converte un punto da coordinate globali a locali del robot

Cinematica di Robot Mobile su Ruote



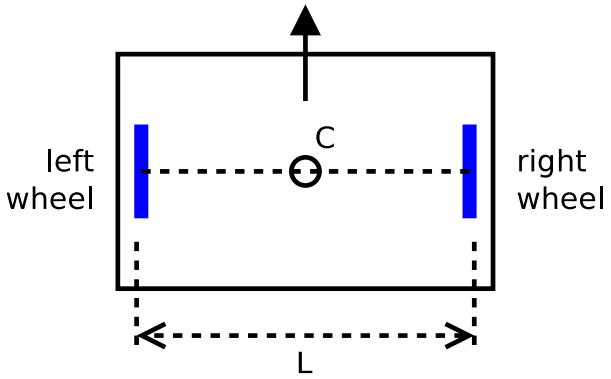
- La **posa** istantanea di un robot su un ambiente **bidimensionale** è caratterizzata da:
  - Posizione *x*
  - Posizione *y*
  - Orientamento *theta* (rispetto all'asse *x*)

Cinematica di Robot Mobile su Ruote



- Si considerano due sistemi di riferimento:
  - **Body system**, sistema di riferimento *locale* del robot (x', y')
  - **Earth system**, sistema di riferimento *globale* dell'ambiente (x, y)
- La trasformazione tra un sistema e l'altro avviene usando le normali formule di roto-traslazione, implementate nei metodi **`global_to_local`** e **`local_to_global`** della classe **`Pose`**

Modello di Robot Mobile a due ruote indipendenti



- Due ruote indipendent poste ai lati del robot
- Il punto C è detto **centro di istantanea rotazione** ed è il riferimento per la posizione globale del robot
- I parametri di geometria per il modello cinematico sono:
  - r_L, raggio della ruota sinistra
  - r_R, raggio della ruota destra
  - n_L, numero di tick per giro dell'encoder sinistro
  - n_R, numero di tick per giro dell'encoder destro
  - L, distanza tra i punti di contatto delle due ruote (**wheelbase**)

Cinematica del Robot Mobile a due ruote indipendenti

- Parametri:
  - r_L, raggio della ruota sinistra
  - r_R, raggio della ruota destra
  - n_L, numero di tick per giro dell'encoder sinistro
  - n_R, numero di tick per giro dell'encoder destro
  - L, distanza tra i punti di contatto delle due ruote (**wheelbase**)
- Detti $\Delta tick_L$ e $\Delta tick_R$ il numero di **tick contati**, su ruota destra e sinistra, nell'intervallo di campionamento ΔT, abbiamo:

$$\Delta d_L = \frac{2\pi r_L}{n_L}\Delta tick_L \qquad \Delta d_R = \frac{2\pi r_R}{n_R}\Delta tick_R$$

$$\Delta\theta = \frac{\Delta d_R - \Delta d_L}{L} \qquad \Delta d = \frac{\Delta d_R + \Delta d_L}{2}$$

$$v_L = \frac{\Delta d_L}{\Delta T} \qquad v_R = \frac{\Delta d_R}{\Delta T}$$

Cinematica del Robot Mobile a due ruote indipendenti

- Detti $\Delta tick_L$ e $\Delta tick_R$ il numero di **tick contati**, su ruota destra e sinistra, nell'intervallo di campionamento ΔT, abbiamo:

$$\Delta d_L = \frac{2\pi r_L}{n_L}\Delta tick_L \qquad \Delta d_R = \frac{2\pi r_R}{n_R}\Delta tick_R$$

$$\Delta\theta = \frac{\Delta d_R - \Delta d_L}{L} \qquad \Delta d = \frac{\Delta d_R + \Delta d_L}{2}$$

$$v_L = \frac{\Delta d_L}{\Delta T} \qquad v_R = \frac{\Delta d_R}{\Delta T}$$

- Dove:
  - Δd_L, Δd_R, distanza percorsa (in ΔT) dalla ruota sinistra/destra
  - $\Delta theta$, variazione dell'orientamento (in radianti)
  - v_L, v_R, velocità istantanea della routa sinistra/destra

Cinematica del Robot Mobile a due ruote indipendenti

$$\Delta d_L = \frac{2\pi r_L}{n_L}\Delta tick_L \qquad \Delta d_R = \frac{2\pi r_R}{n_R}\Delta tick_R$$

$$\Delta\theta = \frac{\Delta d_R - \Delta d_L}{L} \qquad \Delta d = \frac{\Delta d_R + \Delta d_L}{2}$$

$$v_L = \frac{\Delta d_L}{\Delta T} \qquad v_R = \frac{\Delta d_R}{\Delta T}$$

- Sulla base delle relazioni precedenti, l'aggiornamento della posa si ottiene con:

$$\begin{aligned} x &= x + \Delta d \cos(\theta + \frac{\Delta\theta}{2}) \\ y &= y + \Delta d \sin(\theta + \frac{\Delta\theta}{2}) \\ \theta &= \theta + \Delta\theta \end{aligned}$$

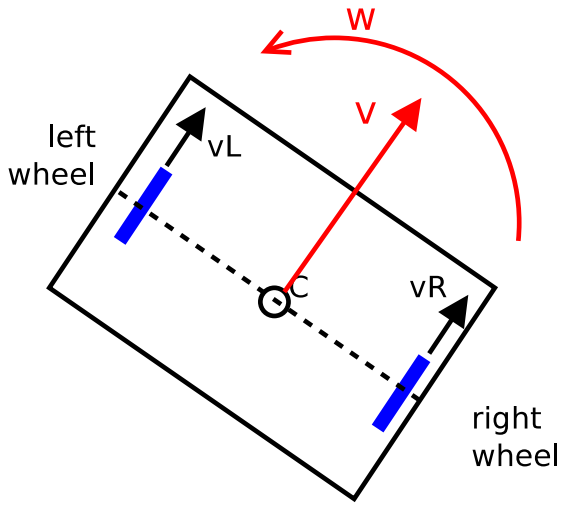
Cinematica del Robot Mobile a due ruote indipendenti

- Sulla base delle relazioni precedenti, l'aggiornamento della posa si ottiene con:

$$\begin{aligned} x &= x + \Delta d \cos(\theta + \frac{\Delta\theta}{2}) \\ y &= y + \Delta d \sin(\theta + \frac{\Delta\theta}{2}) \\ \theta &= \theta + \Delta\theta \end{aligned}$$

- Le relazioni precedenti, note come **formule per l'odometria**, si determinano tramite l'**integrazione dell'equazione del moto**
- Tale integrazione (che non può essere effettuata analiticamente) è effettuata **numericamente** e pertanto contiene un **errore che si propaga in modo additivo** nella stima della posa (x, y, θ)
- La posa stimata (x, y, θ) è pertanto soggetta ad un errore non nullo e non eliminabile che cresce con l'andare del tempo/distanza percorsa

Cinematica del Robot Mobile a due ruote indipendenti



- La **velocità** è invece rappresentata secondo due possibili sistemi di riferimento
  - (v_L, v_R), velocità istantanea delle routa sinistra e destra
  - (v, ω), velocità istantanea di **traslazione** e **rotazione** del punto C
- Tra i due sistemi di riferimento esiste la seguente trasformazione:

$$v = \frac{v_R + v_L}{2} \qquad \omega = \frac{v_R - v_L}{L}$$

$$v_L = v - \frac{L\omega}{2} \qquad v_R = v + \frac{L\omega}{2}$$

Classe Kinematics

- Nel sistema di Motion Control, la classe **`Kinematics`** gestisce la cinematica del robot secondo il modello illustrato
- E' un **task periodico** che viene eseguito ogni 5*ms* (metodo `run()`), e comprende la lettura degli encoder ed il calcolo delle grandezze cinematiche

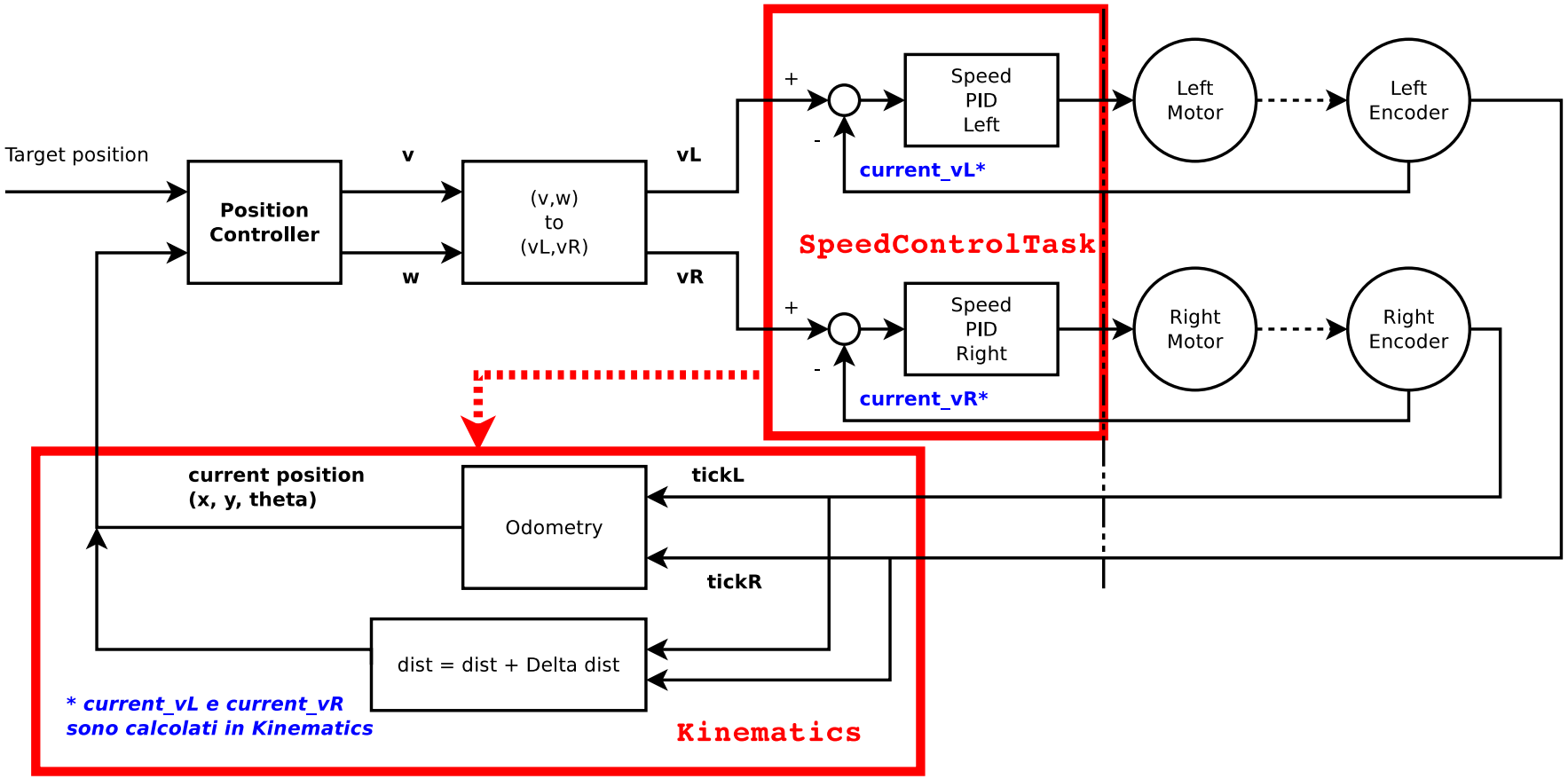
```
class Kinematics : public PeriodicTask {
 public:
    Kinematics(Pose & p,
               float radius_left, float radius_right,
               float wheel_base, float ticks);
    void run();
    float speed_left() { return m_speed_left; };
    float speed_right() { return m_speed_right; };
    Pose & pose() { return robot_pose;};
 private:
    float m_radius_left, m_radius_right;              // raggio ruote
    float m_wheel_factor_left, m_wheel_factor_right;// fattore tick/distanza
    float m_wheelbase;                                // distanza tra le ruote
    float m_ticks_per_revolution;                     // tick/giro encoder
    int m_poscount_left, m_poscount_right;            // distanza percorsa in DT
    Pose & robot_pose;                                // riferimento alla posa del robot
    float m_speed_left, m_speed_right;                // vel sx e dx istantanee
    float m_linear_speed, m_angular_speed;            // vel lineare e angolare
    float m_linear_distance, m_angular_distance;      // distanza lineare e angolare
};
```

Controllo in velocità, classe SpeedControlTask

- La classe **`SpeedControlTask`** implementa i PID per il controllo di velocità delle ruote sinistra e destra
- E' un **task periodico** che viene eseguito ogni 10*ms* (metodo `run()`)
- Essa include un riferimento alla classe **`Kinematics`** in modo da poter conoscere le velocità effettive delle due ruote

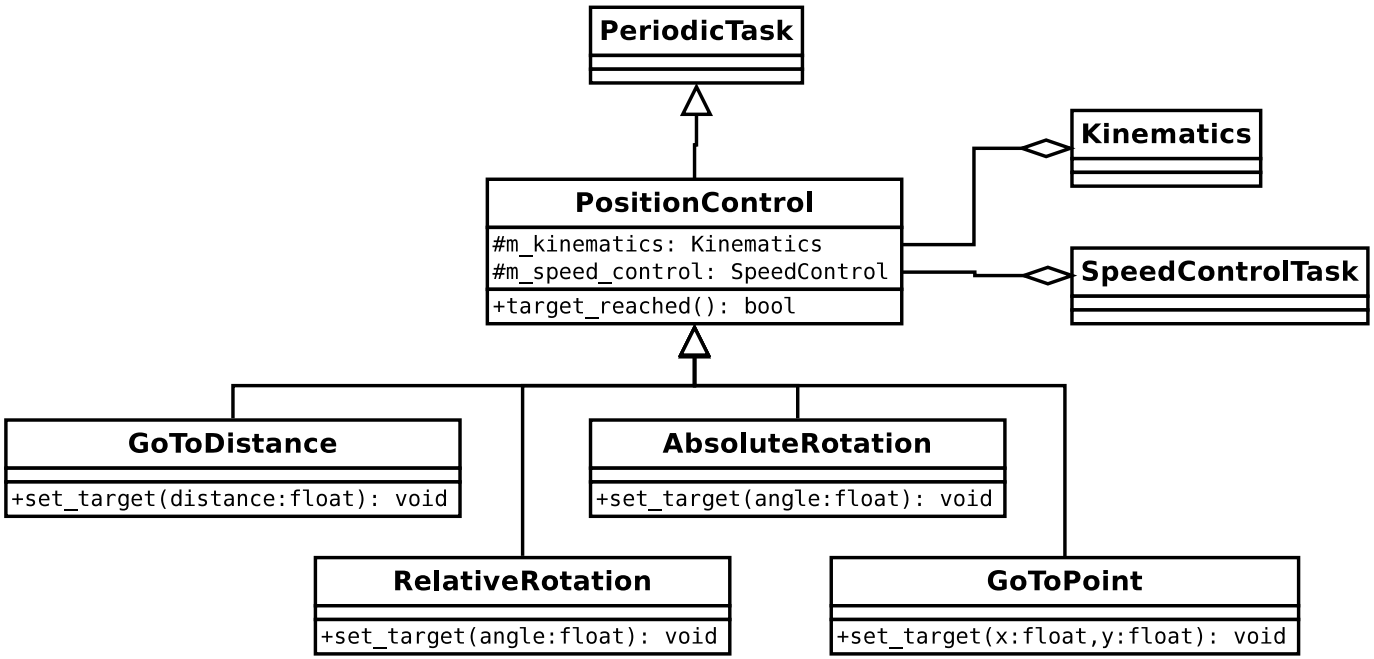
```
class SpeedControlTask : public PeriodicTask {
 public:
    SpeedControlTask(Kinematics & kin);
    void run();
    void off(bool stop_pwm = true);

    // imposta i parametri dei PID
    void set_params(float kp, float ki, float kd);

    // imposta le vel target
    void set_targets(float l, float r);

    // restituisce le vel correnti
    void get_current_speeds(float & l, float & r);
 private:
    Kinematics & m_kinematics;           // riferimento alla cinematica
    int m_pwm_left, m_pwm_right;         // valori del PWM impostati dai PID
    float m_target_left, m_target_right; // vel target
    Pid m_left_pid, m_right_pid;         // PID di ruota sx e dx
};
```

Relazioni tra Schema di Controllo e Oggetti Software

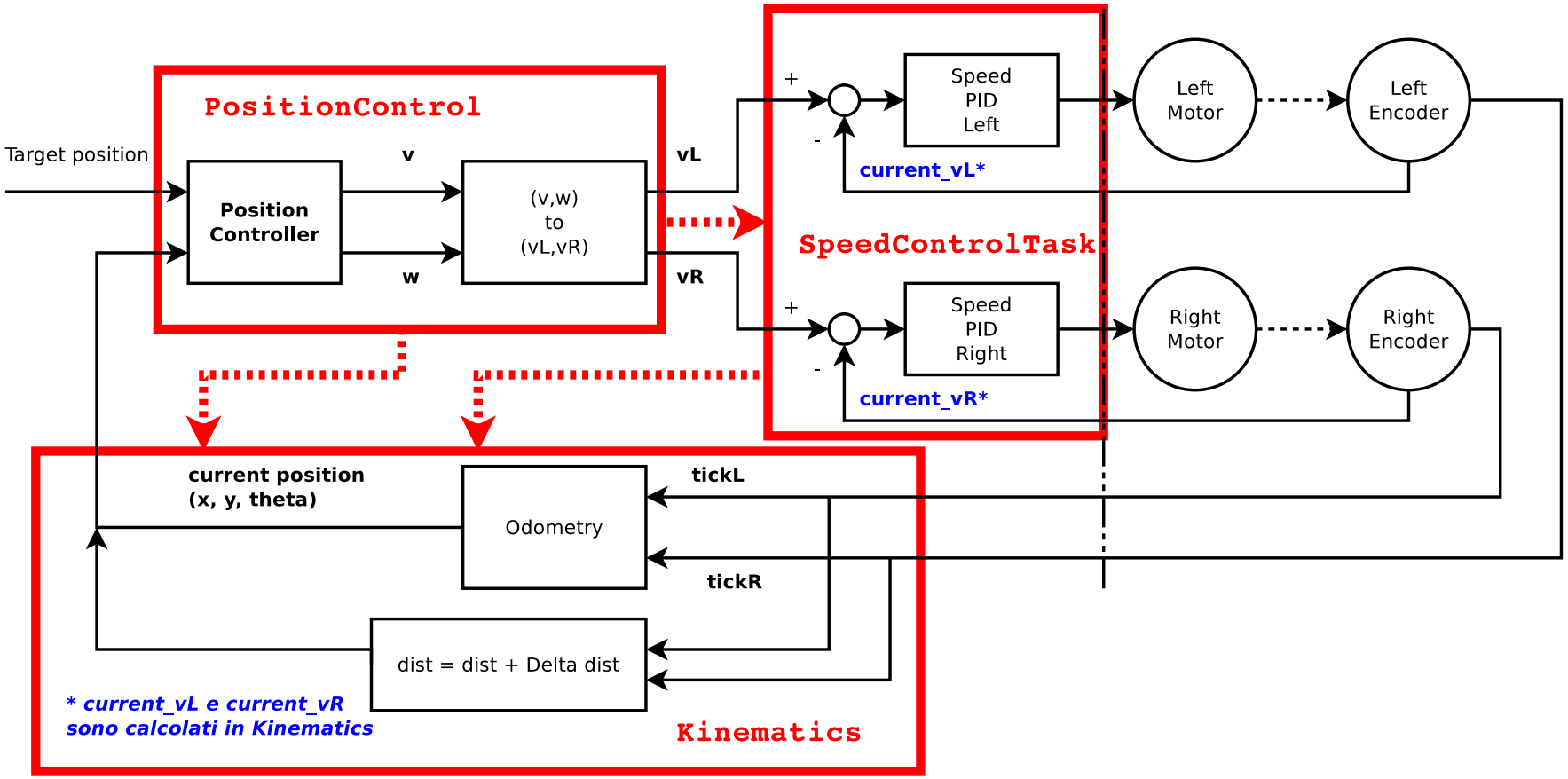


Controllo in Posizione: Schema delle classi

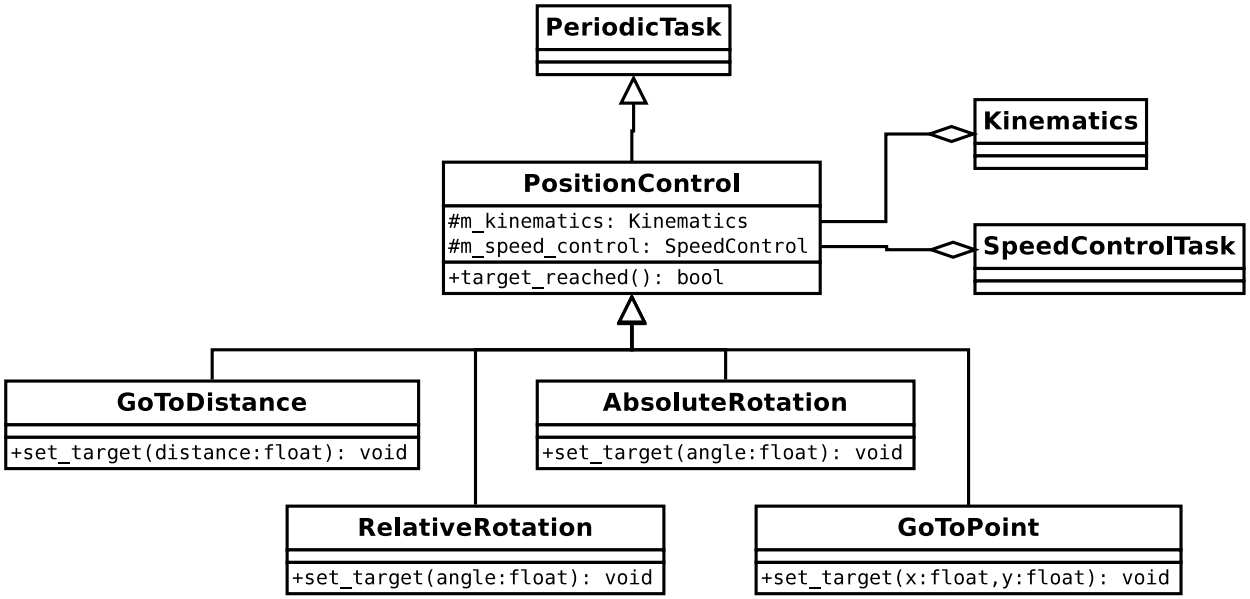
- La classe astratta **`PositionControl`** rappresenta il task periodico che implementa il **generico controllo** in posizione
- Essa include un riferimento all'oggetto **`Kinematics`** ed all'oggetto **`SpeedControlTask`**
- Da **`PositionControl`** vengono derivate le classi che implementano lo **specifico controllo**



Relazioni tra Schema di Controllo e Oggetti Software

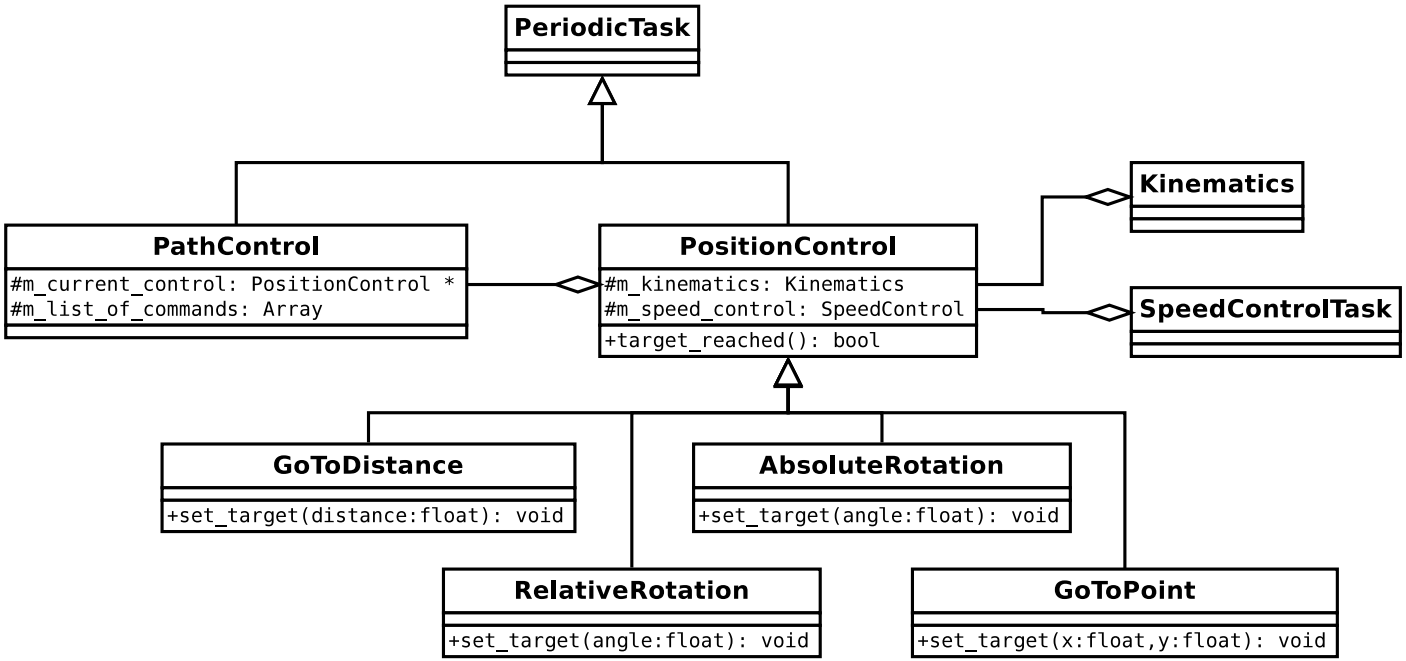


Controllo in Posizione “Intercambiabile”



- Dei vari “`PeriodicTask`” che implementano gli specifici controlli in posizione, **solo uno per volta** può essere attivo
- Ciò viene realizzato andando ad “accendere”/”spegnere” i singoli task periodici
- Ogni task periodico può essere infatti acceso o spento con:
  - **metodo** `void on()`, accende il task
  - **metodo** `void off()`, spegne il task (esso non verrà più schedulato)

Path Control



- La classe **`PathControl`** gestisce una **lista di comandi** di geometria
- Usa un puntatore a **PositionControl** che rappresenta il controllore in posizione corrente
- Per cambiare controllo:
  - `m_current_control->off();`
  - `m_current_control = // nuovo controllo;`
  - `m_current_control->set_target(...);`
  - `m_current_control->on();`

Architettura Software del Motion Control

Corrado Santoro

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

Programmazione Sistemi Robotici