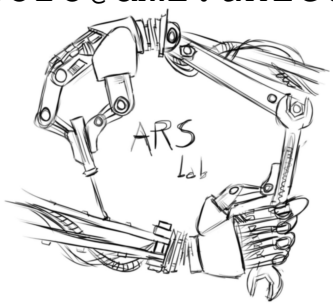
Modellazione, simulazione e controllo di un manipolatore piano

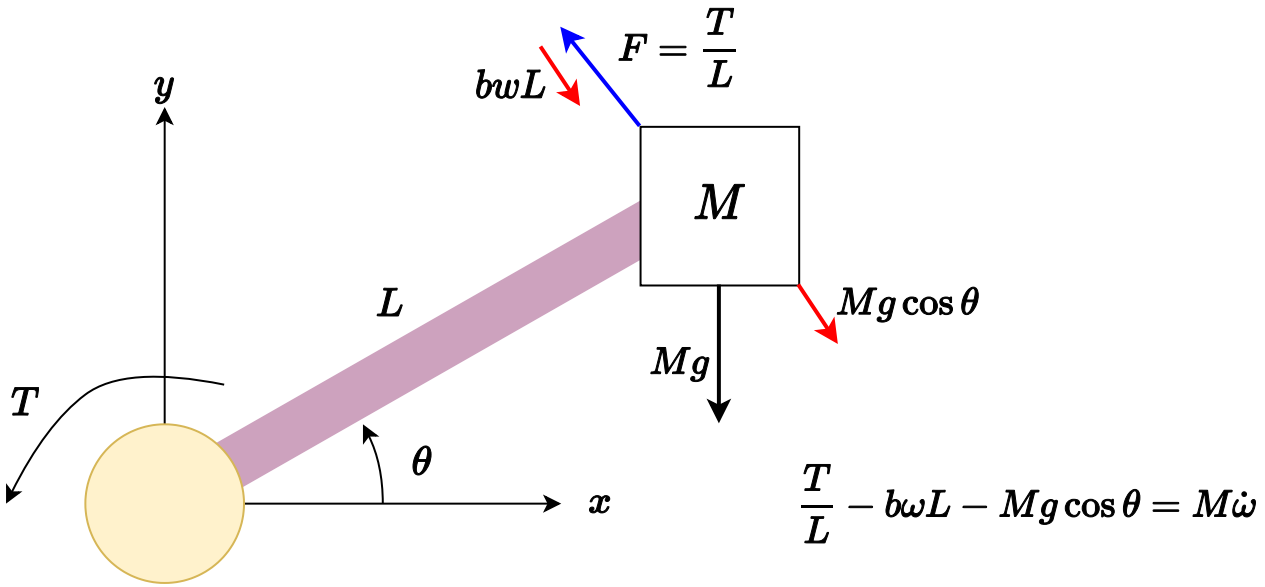
Corrado Santoro

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



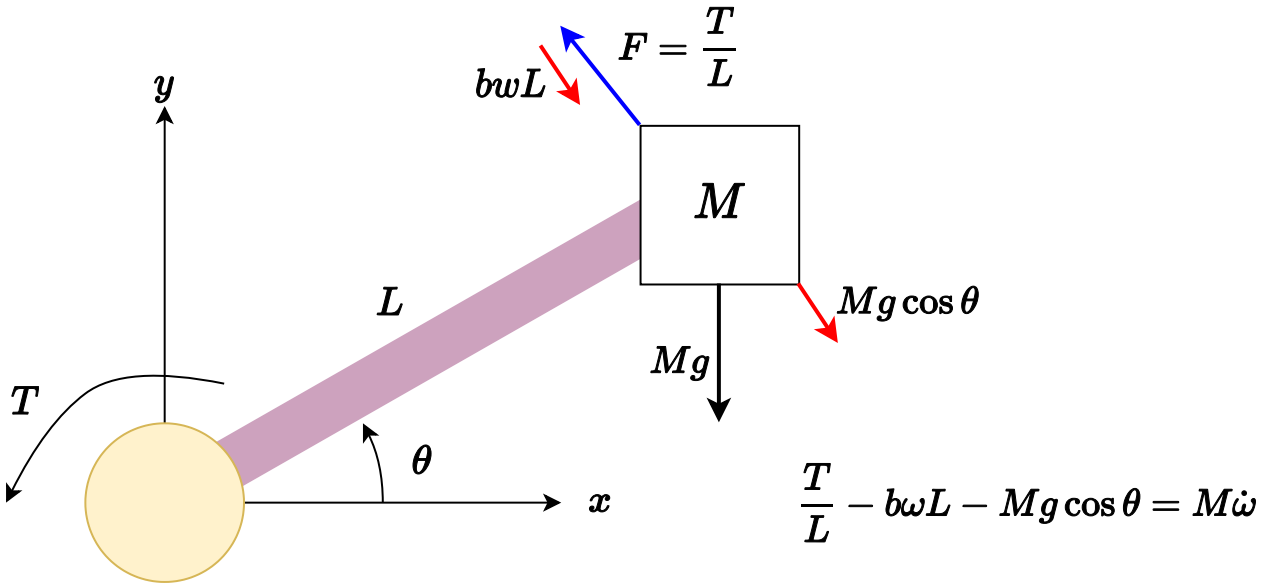
Programmazione Sistemi Robotici

Modello fisico del braccio



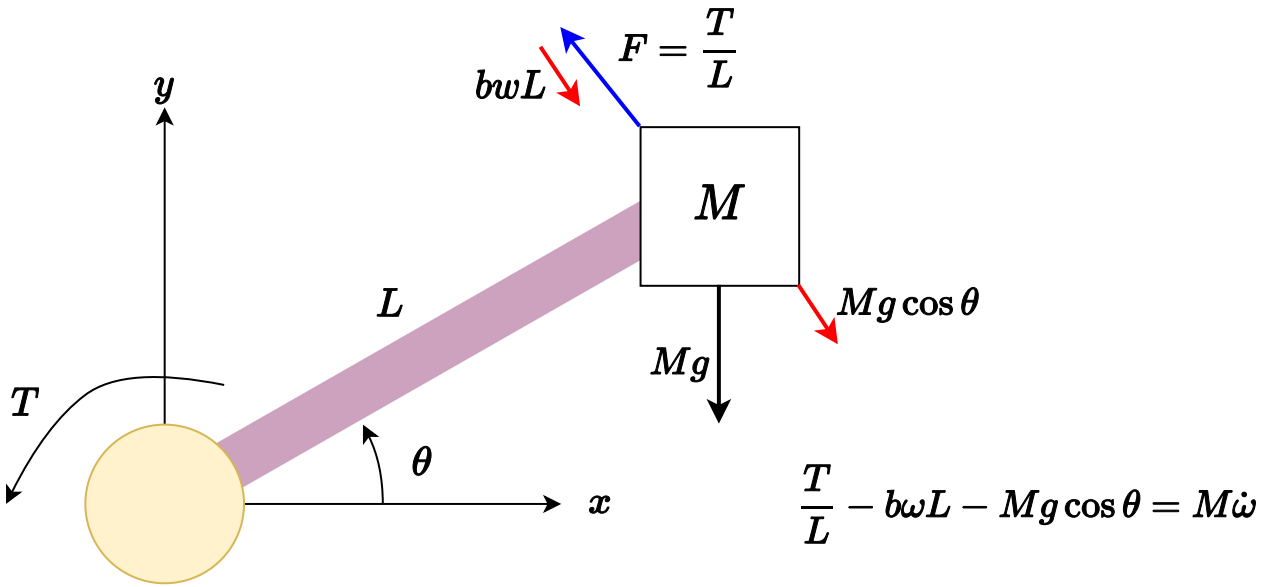
- Uso del sistema di riferimento cartesiano ($\theta = 0$ con braccio orizzontale)
- Utilizzo della **coppia (torque)** come input per la trazione
- Utilizzo della velocià lineare (ωL) per il calcolo della forza di attrito viscoso

Modello fisico del braccio



$$\begin{aligned} \dot{\omega} &= -\frac{bL}{M}\omega - g\cos\theta - \frac{1}{ML}T \\ \dot{\theta} &= \omega \end{aligned}$$

Modello fisico discreto del braccio



$$\omega(k+1) = (1 - \frac{bL}{M}\Delta T)\omega(k) - g\Delta T\cos\theta(k) - \frac{\Delta T}{ML}T(k)$$
$$\theta(k+1) = \theta(k) + \Delta T\omega(k)$$

Implementazione modello fisico

Elenco delle classi

- **`ArmElement`** (file “arm_model.py”), modello fisico del singolo giunto con braccio
- **`ArmControl`** (file “arm_control.py”), controllo in velocità e posizione angolari del singolo giunto
- **`SingleArm`** (file “arm.py”), manipolatore a braccio singolo (modello + sistema di controllo)
- **`TwoJointsArm`** (file “arm.py”), manipolatore a due giunti (modello + sistema di controllo)
- **`SingleArmPainter`** (file “arm_painters.py”), funzioni per disegnare il manipolatore a braccio singolo
- **`TwoJointsArmPainter`** (file “arm_painters.py”), funzioni per disegnare il manipolatore a due giunti
- **`Trajectory`** (file “trajectory.py”), traiettoria retta verso un punto target (x, y)

Implementazione modello fisico

File: arm_model.py

```
import math

GRAVITY = 9.81

class ArmElement:

    def __init__(self, _L, _M, _b):
        self.w = 0
        self.theta = 0
        self.L = _L
        self.M = _M
        self.b = _b

    def evaluate(self, _input_torque, delta_t):
        self.w = self.w - GRAVITY * delta_t * math.cos(self.theta) - \
            (self.b * delta_t * self.w * self. L) / self.M + \
            delta_t * _input_torque / (self.M * self.L)
        self.theta = self.theta + delta_t * self.w

    def get_pose(self):
        return (self.L * math.cos(self.theta), self.L * math.sin(self.theta) )
```

Implementazione modello fisico

File: arm_control.py

```
class ArmControl:

    def __init__(self, arm, use_profile):
        self.arm = arm
        self.use_profile = use_profile
        self.speed_controller = PI_SAT_Controller(20, 10, 10)
        if self.use_profile:
            self.position_controller = ProfilePositionController(6.0, 2.0, 2.0)
        else:
            self.position_controller = PSAT_Controller(12, 8)
        self.target = 0
        self.w_target = 0

    def set_target(self, target):
        self.target = math.radians(target)

    def evaluate(self, delta_t):
        ...
```

Implementazione modello fisico

File: `arm.py`

```
class SingleArm:

    def __init__(self, use_profile):
        self.arm_model = ArmElement(0.1, 0.5, 7e-5)
        self.arm_control = ArmControl(self.arm_model, use_profile)

    def set_target(self, theta1):
        self.arm_control.set_target(theta1)

    def evaluate(self, delta_t):
        self.arm_control.evaluate(delta_t)

    def get_pose(self):
        return [ self.arm_model.get_pose() ]

    def get_angle(self):
        return math.degrees(self.arm_model.theta)
```

Implementazione modello fisico

File: `arm.py`

```
class TwoJointsArm:

    def __init__(self, trajectory_controller, use_profile):
        self.element_1_model = ArmElement(0.2, 0.5 + 0.5, 7e-5)
        self.element_1_control = ArmControl(self.element_1_model, use_profile)
        self.element_2_model = ArmElement(0.1, 0.5, 7e-5)
        self.element_2_control = ArmControl(self.element_2_model, use_profile)
        self.trajectory = trajectory_controller

    def set_target(self, theta1, theta2):
        self.element_1_control.set_target(theta1)
        self.element_2_control.set_target(theta2)

    def evaluate(self, delta_t):
        self.element_1_control.evaluate(delta_t)
        self.element_2_control.evaluate(delta_t)

    def get_pose_degrees(self):
        return (math.degrees(self.element_1_model.theta),
                math.degrees(self.element_2_model.theta) )

    def get_pose(self):
        ...
        return [ (x1, y1), (x2, y2) ]

    def inverse_kinematics(self, xt, yt):
        ...
        return (math.degrees(theta1), math.degrees(theta2))
```

Implementazione modello fisico

File: `arm.py`

```
class TwoJointsArm:

    ....

    def set_target_xy(self, x, y):
        p = self.get_pose()
        self.trajectory.set_target(p[1][0], p[1][1], x, y)

    def evaluate_trajectory(self, delta_t):
        (self.trajectory_x, self.trajectory_y) = self.trajectory.evaluate(delta_t)
        (th1, th2) = self.inverse_kinematics(self.trajectory_x, self.trajectory_y)
        if th1 is not None:
            self.set_target(th1, th2)
        self.evaluate(delta_t)

    def set_target_xy_to_angles(self, xt, yt):
        (th1, th2) = self.inverse_kinematics(xt, yt)
        p = self.get_pose_degrees()
        self.trajectory.set_target(p[0], p[1], th1, th2)

    def evaluate_trajectory_angles(self, delta_t):
        (th1, th2) = self.trajectory.evaluate(delta_t)
        self.set_target(th1, th2)
        self.evaluate(delta_t)
```

Implementazione modello fisico

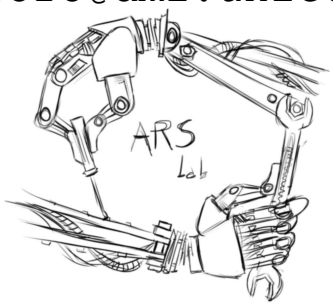
File: `trajectory.py`

```
class Trajectory:

    def __init__(self, vmax, accel, decel):
        ...

    def set_target(self, start_x, start_y, target_x, target_y):
        ...

    def evaluate(self, delta_t):
        if (self.target_pos - self.virtual_pos) < self.decel_distance:
            # fase di decelerazione
            current_accel = -self.decel
        else:
            # fase di accelerazione o moto a vel costance
            current_accel = self.accel

        self.virtual_pos += self.virtual_speed * delta_t + \
          0.5 * current_accel * delta_t * delta_t

        self.virtual_speed += current_accel * delta_t

        if self.virtual_speed >= self.vmax: self.virtual_speed = self.vmax
        if self.virtual_speed <= 0: self.virtual_speed = 0

        vp_x = self.virtual_pos * math.cos(self.target_heading)
        vp_y = self.virtual_pos * math.sin(self.target_heading)
        return (self.start_x + vp_x, self.start_y + vp_y)
```

Modellazione, simulazione e controllo di un manipolatore piano

Corrado Santoro

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



Programmazione Sistemi Robotici