CURSO: SISTEMAS DE ACTUACIÓN EN MECATRÓNICA

Video 03 — Parte 2A: Sistemas Mecánicos — Tipos de Movimiento, Cadenas Cinemáticas y Levas | Parte 2: Sistemas de Actuación Mecánica y Transmisión de Movimiento (Sección A)

1. Síntesis Técnica y Objetivos de Ingeniería

CINEMATICA DE MECANISMOS: MOVILIDAD DE GRUEBLER, CUATRO BARRAS DE GRASHOF Y LEVAS CICLOIDALES

6 Grados de Libertad (6-DoF), Pares Inferiores/Superiores, Inversiones Cinematicas y Perfiles s(theta), v, a, Jerk

- 1. Funciones mecanicas: amplificar fuerza/par, adaptar velocidad/inercia y transformar rotacion en traslacion u oscilacion.
- 2. Tipos de movimiento (traslacion, rotacion, plano general y 6-DoF espaciales) y pares cinematicos inferiores vs superiores.
- 3. Movilidad de Gruebler-Kutzbach GDL = 3(n-1) - 2*j1 - j2 y Ley de Grashof (s + l <= p + q) en el cuadrilatero articulado.
- 4. Levas de disco/cilindricas y seguidores (cuna, rodillo, cara plana): diseno cicloidal con Jerk finito y angulo de presion <= 30 deg.

INVERSIONES DEL MECANISMO DE 4 BARRAS DE GRASHOF (s + l <= p + q)

- `Caso 1: Bancada adyacente al eslabon mas corto (s)`
- `-> Mecanismo MANIVELA-BALANCIN (rotacion a oscilacion)`
- `Caso 2: Bancada ES el eslabon mas corto (s)`
- `-> Mecanismo DOBLE MANIVELA (Drag-Link, 2 giros 360°)`
- `Caso 3: Bancada opuesta al eslabon mas corto (s)`
- `-> Mecanismo DOBLE BALANCIN (ambos oscilan)`
- `Caso 4: Si s + l > p + q (No Grashof)`
- `-> Triple balancin (ningun eslabon da vuelta completa)`

ECUACIONES DEL PERFIL CICLOIDAL DE LEVA (SUBIDA L EN ANGULO BETA)

- Desplazamiento: s(th) = L * [th/beta - sin(2*pi*th/beta)/(2*pi)]
- Velocidad : v(th) = (L*w/beta) * [1 - cos(2*pi*th/beta)]
- Aceleracion : a(th) = (2*pi*L*w^2/beta^2) * sin(2*pi*th/beta)
- Sacudida Jerk : j(th) = (4*pi^2*L*w^3/beta^3) * cos(2*pi*th/beta)
- Angulo Presion: tan(phi) = (ds/dth - e) / (Rb + Rr + s(th)) <= tan(30°)

2. Plano 1: Banco de Potencia y Protoboard (2200x1350)

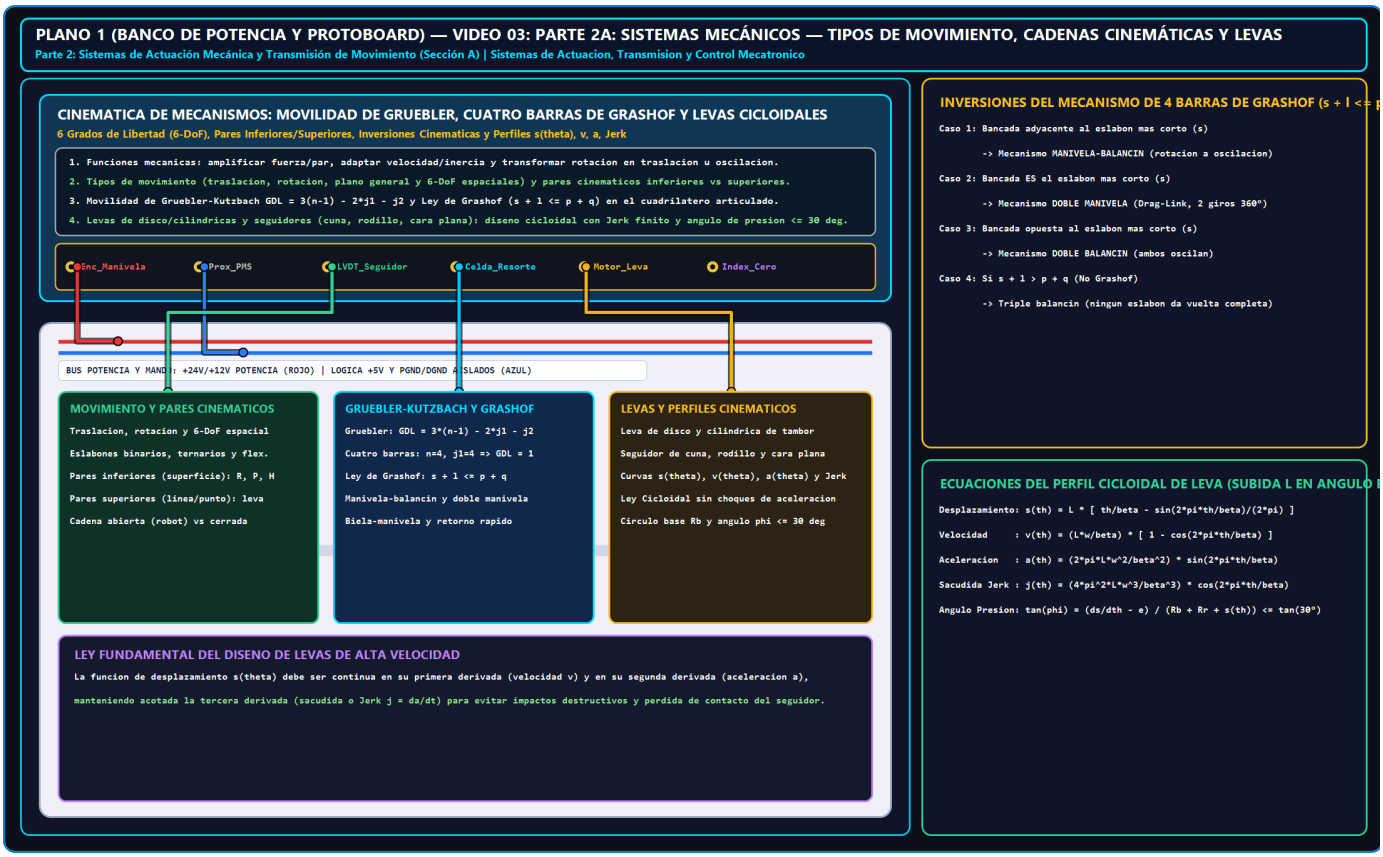


Figura 1: Plano de interconexión de potencia, comando y realimentación para el Video 03.

3. Plano 2: Arquitectura de Placa PCB FR-4 y Flujo de Potencia (2200x1350)



Figura 2: Arquitectura en cuatro etapas, separación de tierras lógica/potencia y criterios de protección.

4. Diagrama 3: Curvas Características y Cinemática (1800x1100)

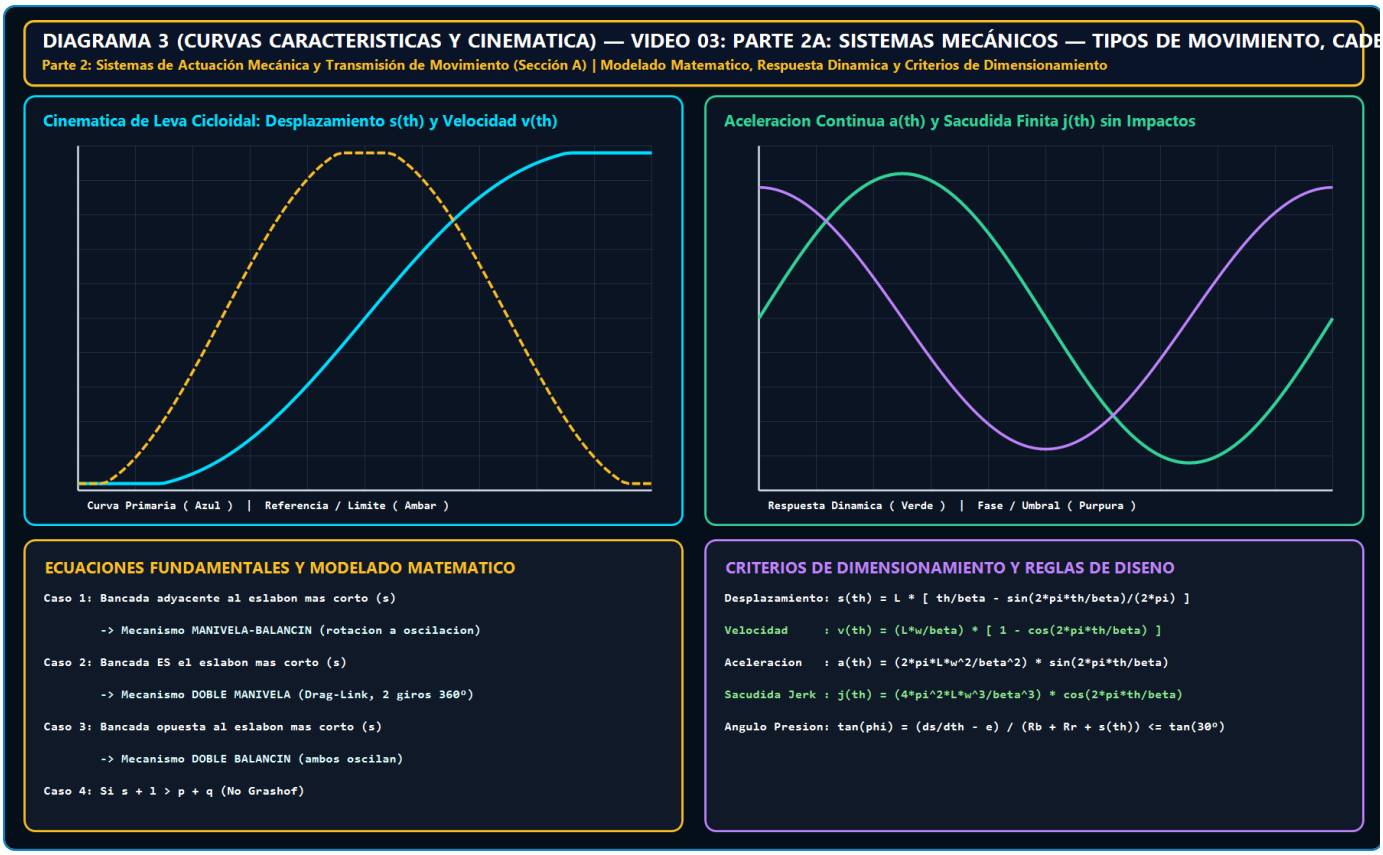


Figura 3: Gráficas de comportamiento dinámico, perfiles cinemáticos y ecuaciones de diseño del Video 03.

5. Firmware de Control de Actuadores en C++ (.ino)

```
// ============================================================================
// CURSO: SISTEMAS DE ACTUACION EN MECATRONICA
// VIDEO 03: Parte 2A: Sistemas Mecánicos – Tipos de Movimiento, Cadenas Cinemáticas y Levas
// Archivo: firmware_actuadores_video_03.ino
// Descripcion: Control determinista de actuadores (electrovalvulas 5/2,
//              solenoide Pick-and-Hold PWM, puente H con rampa, motor paso a
//              paso y lazos en cascada de posicion/velocidad/corriente).
// ============================================================================

#include <Arduino.h>

// Definicion de pines de potencia y sensores de realimentacion
const int PIN_VALVULA_5_2_Y1  = 2;   // Solenoide avance cilindro A+
const int PIN_VALVULA_5_2_Y2  = 3;   // Solenoide retroceso cilindro A-
const int PIN_STEP_DRIVER     = 4;   // Pulso STEP para motor paso a paso
const int PIN_DIR_DRIVER      = 5;   // Direccion DIR motor paso a paso
const int PIN_PWM_SOLENOIDE   = 6;   // Salida PWM Pick-and-Hold
const int PIN_FRENO_EJE_Z     = 7;   // Freno Fail-Safe 24V DC
const int PIN_PWM_PUENTE_H_A  = 9;   // Canal A Puente H (20 kHz / PWM)
const int PIN_PWM_PUENTE_H_B  = 10;  // Canal B Puente H
const int PIN_CORRIente_SHUNT = A0;  // Medicion de corriente de armadura Ia

// Parametros de control cascada y rampa trapezoidal
float consignaPosicionRad     = 3.14159f; // 180 grados
float posicionActualRad       = 0.0f;
float velocidadActualRadS     = 0.0f;
const float KP_POS            = 18.0f;
const float KV_FF             = 0.95f;
const float KP_VEL            = 2.40f;
const float KI_VEL            = 15.0f;
float integradorVel           = 0.0f;

// Funcion de excitacion inteligente Pick-and-Hold para solenoide lineal
void accionarSolenoidePickAndHold(bool activar) {
  static bool estadoPrevio = false;
  static unsigned long tEncendido = 0;

  if (!activar) {
    analogWrite(PIN_PWM_SOLENOIDE, 0);
    estadoPrevio = false;
    return;
  }
  if (!estadoPrevio) {
    // Fase PICK: 100% de ciclo de trabajo (255) durante 40 ms para vencer entrehierro g
    analogWrite(PIN_PWM_SOLENOIDE, 255);
    tEncendido = millis();
    estadoPrevio = true;
  } else if (millis() - tEncendido >= 40) {
    // Fase HOLD: 30% de ciclo de trabajo (77/255) -> reduce 91% la potencia I^2*R
    analogWrite(PIN_PWM_SOLENOIDE, 77);
  }
}

// Lazo en cascada Posicion (P + Feedforward) -> Velocidad (PI) -> PWM Puente H
int ejecutarControlCascadaServo(float posRef, float velRefFF, float posMed, float velMed, float dt) {
  float errorPos = posRef - posMed;
  float cmdVel   = (KP_POS * errorPos) + (KV_FF * velRefFF);

  float errorVel = cmdVel - velMed;
  integradorVel  = constrain(integradorVel + errorVel * dt, -12.0f, 12.0f);
  float cmdParIq = (KP_VEL * errorVel) + (KI_VEL * integradorVel);

  int pwmOut = (int)constrain(cmdParIq * 25.0f, -255.0f, 255.0f);
  if (pwmOut >= 0) {
```

```
    analogWrite(PIN_PWM_PUENTE_H_A, pwmOut);
    analogWrite(PIN_PWM_PUENTE_H_B, 0);
  } else {
    analogWrite(PIN_PWM_PUENTE_H_A, 0);
    analogWrite(PIN_PWM_PUENTE_H_B, -pwmOut);
  }
  return pwmOut;
}

void setup() {
  Serial.begin(115200);
  pinMode(PIN_VALVULA_5_2_Y1, OUTPUT);
  pinMode(PIN_VALVULA_5_2_Y2, OUTPUT);
  pinMode(PIN_STEP_DRIVER, OUTPUT);
  pinMode(PIN_DIR_DRIVER, OUTPUT);
  pinMode(PIN_PWM_SOLENOIDE, OUTPUT);
  pinMode(PIN_FRENO_EJE_Z, OUTPUT);
  pinMode(PIN_PWM_PUENTE_H_A, OUTPUT);
  pinMode(PIN_PWM_PUENTE_H_B, OUTPUT);

  // Secuencia segura de liberacion de freno vertical Z
  digitalWrite(PIN_FRENO_EJE_Z, HIGH);
}

void loop() {
  static unsigned long tPrev = 0;
  unsigned long tNow = millis();

  // Periodo de muestreo de control Ts = 5 ms (200 Hz)
  if (tNow - tPrev >= 5) {
    float dt = (tNow - tPrev) / 1000.0f;
    tPrev = tNow;

    accionarSolenoidePickAndHold(true);
    int pwmAplicado = ejecutarControlCascadaServo(consignaPosicionRad, 0.0f, posicionActualRad,
velocidadActualRadS, dt);

    // Simulacion cinematica de planta de primer orden para telemetria
    velocidadActualRadS += (pwmAplicado * 0.04f - velocidadActualRadS * 0.8f) * dt;
    posicionActualRad   += velocidadActualRadS * dt;

    Serial.print("{\"video\":03,\"pos_rad\":");
    Serial.print(posicionActualRad, 4);
    Serial.print(",\"vel_rads\":");
    Serial.print(velocidadActualRadS, 3);
    Serial.print(",\"pwm\":");
    Serial.print(pwmAplicado);
    Serial.println("}");
  }
}
```

6. Script de Dimensionamiento, Cinemática e Inercia en Python (.py)

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
================================================================================
CURSO: SISTEMAS DE ACTUACION EN MECATRONICA
VIDEO 03: Parte 2A: Sistemas Mecánicos — Tipos de Movimiento, Cadenas Cinemáticas y Levas
Archivo: dimensionamiento_y_cinematica_video_03.py
Descripcion: Herramienta de ingenieria para calculo de cilindros neumaticos/
             hidraulicos, carga critica de pandeo de Euler, valvulas Cv/Kv,
             leva cicloidal, trenes planetarios (Willis), vida de rodamientos
             L10, inercia reflejada y par continuo eficaz T_rms.
================================================================================
"""

import math
from typing import Dict, List

def dimensionar_cilindro_y_euler(
    diam_embolo_mm: float,
    diam_vastago_mm: float,
    presion_bar: float,
    longitud_libre_mm: float,
    modulo_young_gpa: float = 210.0,
    factor_fr: float = 0.10
) -> Dict[str, float]:
    """Calcula fuerzas de avance/retroceso y carga critica de pandeo de Euler."""
    d_m = diam_embolo_mm / 1000.0
    dv_m = diam_vastago_mm / 1000.0
    p_pa = presion_bar * 1e5

    a1 = math.pi * (d_m ** 2) / 4.0
    a2 = math.pi * (d_m ** 2 - dv_m ** 2) / 4.0

    f_av_real = p_pa * a1 * (1.0 - factor_fr)
    f_ret_real = p_pa * a2 * (1.0 - factor_fr)

    # Momento de inercia del vastago I = pi * d^4 / 64 y carga de Euler F_K
    inercia_i = (math.pi * (dv_m ** 4)) / 64.0
    e_pa = modulo_young_gpa * 1e9
    lk_m = longitud_libre_mm / 1000.0
    f_euler = (math.pi ** 2) * e_pa * inercia_i / (lk_m ** 2)

    return {
        "fuerza_avance_N": round(f_av_real, 2),
        "fuerza_retroceso_N": round(f_ret_real, 2),
        "carga_critica_euler_N": round(f_euler, 2),
        "factor_seguridad_pandeo": round(f_euler / max(1.0, f_av_real), 2)
    }

def cinematica_leva_cicloidal(elevacion_mm: float, beta_rad: float, omega_rads: float, theta_rad: float) ->
Dict[str, float]:
    """Evalua desplazamiento s, velocidad v, aceleracion a y sacudida Jerk en leva cicloidal."""
    u = theta_rad / beta_rad
    s = elevacion_mm * (u - math.sin(2.0 * math.pi * u) / (2.0 * math.pi))
    v = (elevacion_mm * omega_rads / beta_rad) * (1.0 - math.cos(2.0 * math.pi * u))
    a = (2.0 * math.pi * elevacion_mm * (omega_rads ** 2) / (beta_rad ** 2)) * math.sin(2.0 * math.pi * u)
    j = (4.0 * (math.pi ** 2) * elevacion_mm * (omega_rads ** 3) / (beta_rad ** 3)) * math.cos(2.0 * math.pi *
u)
    return {"s_mm": round(s, 4), "v_mms": round(v, 3), "a_mms2": round(a, 2), "jerk_mms3": round(j, 1)}

def dimensionar_servomotor_inercia_y_rms(
    masa_carga_kg: float,
    paso_husillo_mm: float,
    reduccion_i: float,
```

```
    eficiencia_eta: float,
    inercia_motor_kgm2: float,
    aceleracion_rads2: float,
    par_carga_const_nm: float,
    tiempos_s: List[float]
) -> Dict[str, float]:
    """Calcula la inercia reflejada al eje del motor, el par pico y el par eficaz T_rms."""
    p_m = paso_husillo_mm / 1000.0
    j_carga = masa_carga_kg * ((p_m / (2.0 * math.pi)) ** 2)
    j_ref = j_carga / ((reduccion_i ** 2) * eficiencia_eta)
    ratio_inercia = j_ref / inercia_motor_kgm2

    t_acel = (inercia_motor_kgm2 + j_ref) * aceleracion_rads2
    t1_pico = t_acel + par_carga_const_nm
    t2_const = par_carga_const_nm
    t3_fren = par_carga_const_nm - t_acel
    t4_reposo = 0.0

    t_tot = sum(tiempos_s)
    t_rms = math.sqrt(
        (t1_pico**2 * tiempos_s[0] + t2_const**2 * tiempos_s[1] + t3_fren**2 * tiempos_s[2] + t4_reposo**2 *
tiempos_s[3]) / t_tot
    )
    return {
        "J_reflejada_kgm2": round(j_ref, 7),
        "relacion_inercia_RJ": round(ratio_inercia, 2),
        "par_pico_Nm": round(t1_pico, 3),
        "par_eficaz_Trms_Nm": round(t_rms, 3)
    }

if __name__ == "__main__":
    print("=== REPORTE DE ACTUACION MECATRONICA - VIDEO 03: Parte 2A: Sistemas Mecánicos — Tipos de
Movimiento, Cadenas Cinemáticas y Levas ===")
    print("Cilindro 50/20 mm a 6 bar:", dimensionar_cilindro_y_euler(50.0, 20.0, 6.0, 350.0))
    print("Leva Cicloidal (L=25mm, th=45 deg):", cinematica_leva_cicloidal(25.0, math.pi / 2.0, 10.0, math.pi
/ 4.0))
    print("Dimensionamiento Servo:", dimensionar_servomotor_inercia_y_rms(25.0, 10.0, 3.0, 0.90, 4.5e-5,
1200.0, 0.25, [0.15, 0.20, 0.15, 0.20]))
```