Mangoneado - sistema de etiquetado

Luis Vergara Arellano

Las entidades representadas en mi proyecto son: el sistema de visión, el servidor de robots y los brazos robóticos individuales. Debido a la dependencia entre eventos (detección de posiciones) y acciones (etiquetado) consideré que la mejor solución debería fundamentarse en comunicación asíncrona mediante sockets TCP y concurrencia con hilos POSIX. Los sockets desacoplan completamente ambos sistemas permitiendo que operen como programas independientes. El esquema de sincronización debe evitar que dos robots intenten etiquetar el mismo mango mediante mutexes individuales por mango. Implementé redistribución dinámica de zonas ante fallos de robots. El diagrama de mi solución se muestra en la ilustración 1.

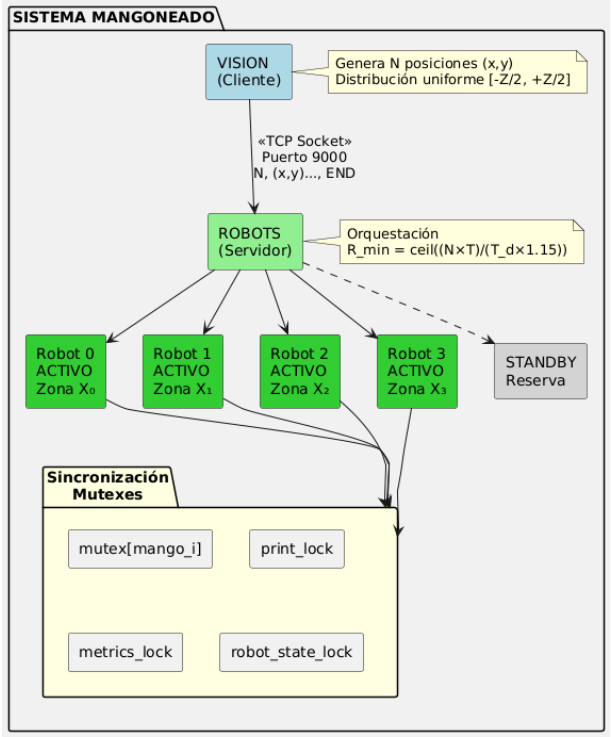

Ilustración 1 - Diagrama de solución

Explicación:

Se crean dos procesos principales.

El proceso **Vision (Cliente)** ejecuta las siguientes tareas:

1. Valida parámetros (IP, puerto, Z, N, seed),
2. Genera N posiciones aleatorias con distribución uniforme en [-Z/2, +Z/2] cm,
3. Se conecta al servidor mediante socket TCP,
4. Envía mensaje estructurado: número N, posiciones (x,y) y token "END",
5. Cierra conexión. Este diseño permite reemplazar la simulación con visión real sin modificar el servidor.

Mangoneado - sistema de etiquetado

Luis Vergara Arellano

El proceso **Robots (Servidor)** ejecuta:

1. Valida parámetros y define manejadores de señales,
2. Acepta conexión TCP y recibe posiciones,
3. Calcula R_min = ceil((N×T_etiqueta)/(T_disponible×1.15)) donde 1.15 da margen del 15%,
4. Crea R hilos, activando solo R_min en modo ACTIVO, el resto en STANDBY,
5. Asigna zonas del eje X: cada robot procesa [(-Z/2 + i×zone_width), (-Z/2 + (i+1)×zone_width)] con zone_width = Z/active_count,
6. Cada hilo ejecuta bucle: verifica mangos en zona, trylock(mutex[mango]), etiqueta, unlock,
7. Simula fallos con Poisson (λ=B×Δt), invocando redistribute_zones() para reasignar espacio,
8. Muestra estado y métricas continuamente.

Con respecto a los tipos de datos, cada mango tiene un mutex individual proporcionando flexibilidad para etiquetado concurrente sin bloqueos. Los mutexes usan pthread con patrón try-lock evitando esperas activas. Los esquemas implementados son:

1. **Activación dinámica:** solo se activan robots necesarios según carga (ejemplo: N=30, T=200ms, disponible=10s requiere R_min=1, resto en STANDBY),
2. **Redistribución ante fallos:** cuando falla un robot (failed=1), redistribute_zones() recalcula zonas entre operativos (ejemplo: con 2 activos y Z=30cm, si falla Robot 0, Robot 1 expande de [0,+15) a [-15,+15) cm),
3. **Granularidad fina:** cada mango con mutex propio permite etiquetado paralelo sin interferencia.

Limitaciones del proyecto: No se implementó priorización de mangos, balanceo dinámico durante ejecución normal (redistribución solo ante fallos), ni métricas de calidad. Robots en STANDBY no se activan automáticamente; robots activos expanden zonas.

Para compilar ejecutar make, y para probar en terminales separadas:

- Terminal A: ./build/robots 9000 4 10 30 200 200 0.05
- Terminal B: ./build/vision 127.0.0.1 9000 30 30 1234

Donde

- robots recibe: puerto, R (robots totales), N (mangos), Z (ancho caja cm), W (ancho banda cm), X (velocidad cm/s), B (prob. fallo/s).
- Vision recibe: IP, puerto, Z, N, seed.