Daniel Santiago Leon Prieto 20221005014

ROSMeetup Bogotá 2025: activismo, comunidad y el estado técnico de la robótica con ROS 2 en Colombia

Como estudiante de quinto semestre de Ingeniería Electrónica de la Universidad Distrital, asistir —aunque sea de forma remota— al ROSMeetup Bogotá 2025 me dejó dos impresiones complementarias. La primera es social: un capítulo que "había estado inactivo" se reactivó gracias a activismo sostenido y hoy es una comunidad articulada, con apoyo de RAS y profesores locales e internacionales. La segunda es técnica: los contenidos presentados —desde la integración de 3D Slicer con ROS 2, hasta control de humanoides, herramientas educativas tipo RosBlocks, robótica de campo para agricultura y la arquitectura modular de un exápodo— muestran que ROS 2 ya no es solo middleware de investigación (intermediario), sino una plataforma madura para construir sistemas reales, distribuidos y seguros. En este ensayo me concentraré en esa segunda capa: ¿qué decisiones de arquitectura, patrones de programación y cadenas de simulación–control permiten que estas demostraciones funcionen?, ¿qué implicaciones tienen para nuestra formación y para proyectos en la Distrital?

El evento arranca con una nota de activación: el capítulo, ahora anclado en la Facultad Tecnológica de la Universidad Distrital, vuelve a estar "fuerte y organizado". La robótica moderna no es monolítica: integra sensores heterogéneos, cómputo en el borde, simulación, control en tiempo (casi) real, visualización y despliegues repetibles. Ninguna persona aísla todas esas capas con excelencia; la comunidad es el mecanismo para cerrar brechas: mantiene repositorios, comparte paquetes, valida controladores, reporta errores (bugs) y, sobre todo, crea estándares de facto (nombres de tópicos, esquemas de marcos TF, políticas de Calidad de Servicio (QoS) en DDS, etc.). El agradecimiento a RAS y la mención de profesores como Mohan Tribedi apuntan a esa red: el valor está en las conexiones que, por ejemplo, facilitan traer ponentes que trabajan en NVIDIA o en universidades que empujan herramientas de simulación quirúrgica. En ese sentido, hasta la referencia a los TurtulBots (la comunidad suele usar TurtleBots como plataforma didáctica) es significativa: son la bisagra entre lo pedagógico y lo real, donde ROS deja de ser teoría y se convierte en tópicos que publican y servicios que responden.

La charla de Arvind Escumar presenta 3D Slicer 2 —un entorno de software para robótica médica e intervenciones mínimamente invasivas— y la tesis técnica central: "hacer que Slicer y ROS se comuniquen" tratando a Slicer como un nodo de ROS. Esa decisión, que suena obvia, es profunda porque fija el patrón de acoplamiento. En lugar de construir un puente (bridge) ad hoc que convierta estructuras internas de Slicer a mensajes ROS, se elige el modelo idiomático: publicadores, suscriptores, parámetros, servicios y acciones de ROS como primitivas universales.

Slicer expone estructuras de datos (escenas, volúmenes, modelos) y observadores que notifican cambios. El acierto es mapear esos eventos a funciones de retorno (callbacks) que publican mensajes ROS en tópicos bien definidos. Por ejemplo, si un módulo de Slicer actualiza una malla 3D (p. ej., reconstrucción de tejido), un observador dispara un publicador rclcpp::Publisher<visualization_msgs::MarkerArray> (o

sensor_msgs/PointCloud2 si corresponde). Inversamente, un suscriptor ROS podría modificar un “nodo” de Slicer al recibir comandos de un planificador externo.

En ROS 2, los parámetros son clave para ajuste en caliente (tuning): ganancias de control, límites de velocidad, indicadores (flags) de depuración. Exponerlos desde Slicer con un cliente de parámetros (parameter client) evita recompilar: basta un
ros2 param set slicer_node safety_margin 3.0.
Los servicios agregan semántica transaccional: “calibra”, “guarda escena”, “carga robot”, “registra imagen”. Ese par (parámetros + servicios) crea una API estable, versionable y comprobable con ros2 test o pruebas de lanzamiento (launch tests).

En aplicaciones médicas, latencia y confiabilidad importan. ROS 2, soportado por DDS, permite perfilar QoS: confiable (reliable) vs mejor esfuerzo (best-effort), conservar-último (keep-last) vs conservar-todo (keep-all), plazos (deadlines) y vida útil (lifespan). Integrar Slicer como nodo exige pensar esas políticas: por ejemplo, seguimiento (tracking) de herramientas quirúrgicas puede ir con reliable + keep-last(1) y plazo corto; la visualización volumétrica remota tolera best-effort. La charla sugiere que este acoplamiento ya está maduro: “la integración y desarrollo han mejorado significativamente frente a años anteriores”.

El ejemplo del tumor de lengua es didáctico: se define una trayectoria en Slicer, se marcan regiones prohibidas (zonas de riesgo) y se provee retroalimentación háptica si el robot se acerca demasiado. Desde ROS, esto se modela bien con una tubería (pipeline): (i) nodo de planificación que genera puntos de paso (waypoints), (ii) nodo de verificación de colisión con la malla del tejido y máscaras binarias de la zona restringida (no-go zone), (iii) nodo de control de baja latencia que envía referencias a actuadores, y (iv) nodo háptico que cierra el lazo con un controlador de dispositivo. La frontera Slicer↔ROS mueve volúmenes, modelos y poses; ROS ejecuta control y seguridad con TF2 y mensajes como geometry_msgs/PoseStamped.

Para quienes estamos en semestres medios, la lección es doble: aprender ROS 2 no es solo rclcpp; es diseñar interfaces (tópicos/servicios) centradas en datos y latencias, y disciplinar la escena (marcos TF consistentes, URDF correcto, políticas de QoS explícitas).

La presentación de Gabriel Díaz aborda un clásico difícil: controlar la postura y la locomoción de humanoides. Desde la ingeniería, esto implica dinámicas acopladas, múltiples grados de libertad y restricciones unilaterales (contacto pie–suelo). Las nociones de estabilidad (ZMP, capturability), coordinación interarticular y elección de controladores condicionan el rendimiento.

Para tareas de postura, un PD bien sintonizado puede bastar; para locomoción y rechazo de perturbaciones, se suelen combinar control de torso, regulador de ZMP y control de trayectoria de los pies. Aparecen enfoques como LQR sobre modelos linealizados o control de espacio operacional (operational space control) con jerarquías de tareas. El mensaje del ponente es pragmático: la elección del controlador afecta la estabilidad observada en

simulación. Esto nos recuerda que la robótica real no es “elegir un algoritmo”: es resolver la tríada modelo–sensor–actuador con limitaciones de fricción, saturaciones y retardo.

La simulación, mostrada con ejemplos, desacopla el riesgo de aprendizaje. Con ROS 2, lo típico es montar un lazo: simulador físico → publicadores de sensor_msgs → estimador de estado (EKF/UKF) → controlador → comandos trajectory_msgs/JointTrajectory o geometry_msgs/Twist. La validez experimental depende de la fidelidad del modelo (inerciales, rigideces, contacto), pero incluso con imperfecciones, permite comparar controladores y hacer barridos de parámetros. En la Distrital, tener escenarios reproducibles con Gazebo/Ignition, ros2_control, y pruebas automatizadas en CI podría ser un hito: reduce el “en mi equipo sí funcionaba”.

Juan Andrés Ramírez presenta RosBlocks, inspirado en Scratch, para permitir que principiantes “creen robots” sin programar en C++/Python. Desde software, esto implica bloques visuales que generan nodos, publicadores y suscriptores preconfigurados, probablemente usando rosbridge o envoltorios (wrappers) que traduzcan acciones gráficas a mensajes ROS. El reto no es solo experiencia de usuario (UX): es definir contratos de datos seguros. Si un bloque “publica velocidad”, ¿qué marco usa?, ¿qué límites valida?, ¿qué QoS aplica?, ¿qué ocurre si el suscriptor no existe? RosBlocks parece orientarse a educación juvenil, con enfoque constructivista: el estudiante manipula objetos (bloques) y ve efectos inmediatos en un robot simulado o físico. Esto cierra un vacío real: muchos cursos presentan ROS como “instale esto y corra un launch”. RosBlocks propone un peldaño intermedio donde el concepto (tópico, servicio, frecuencia rate) se interioriza antes de pelear con colcon y dependencias.

RosBlocks sirve para prototipado rápido de tuberías (pipelines) y para docencia inversa: primero se hace funcionar con bloques; luego se inspecciona el código generado para enseñar mejores prácticas (nombres de tópicos, manejo de excepciones, temporizadores). Integrarlo en semilleros puede aumentar retención y transferencia de conocimiento.

Un exápodo típico usa patrones de marcha como tripod, ripple o wave. Cada pata tiene cinemática inversa y el controlador coordina fases para mantener estabilidad estática o cuasiestática. En ROS 2, esto suele implementarse con un planificador de apoyos, un generador de trayectorias para pies y un adaptador de terreno que ajusta altura y orientación según sensores (IMU, LIDAR, profundidad).

La mención a “arquitectura modular basada en ROS 2” sugiere nodos como: gait_planner, leg_ik, state_estimator, foot_contact_detector, terrain_mapper, command_mux y hw_interface (vía ros2_control). Los tópicos clave incluyen cmd_vel, joint_states, imu/data, pointcloud, foot_contacts, y tf/tf_static. Las acciones pueden manejar trayectorias largas, y los parámetros permiten cambiar marcha en tiempo de ejecución.

El equipo reporta pruebas y resultados, además de “desafíos en entornos reales”. Esa frase, en robótica de piernas, suele traducirse a tres realidades: (i) incertidumbre del terreno, (ii) derivas en estimación del estado, y (iii) holguras mecánicas que rompen supuestos del modelo. La salida es iterativa: fusionar sensores (EKF entre IMU–odometría–visual),

recalibrar, y cerrar el lazo con mapas de elevación. Es valiosa la transparencia: simular es necesario, pero solo el campo descubre resonancias, vibraciones y microgolpes que tumban controladores “bonitos”.

La investigación de Edna Carolina Moriones sobre robótica agrícola se centra en navegación autónoma en ambientes irregulares. Técnicamente, esto obliga a salir de pasillos planos con marcadores fiduciales y entrar a campos sin geometría predecible.

Un stack típico incluye LIDAR (detección de obstáculos y surcos), IMU (actitud), GPS RTK (cuando hay cielo abierto y se requiere precisión centimétrica), y cámara RGB/Depth para filas de cultivo. El filtro de Kalman extendido (EKF) —o variantes— fusiona estas mediciones para estimar pose. En ROS 2, paquetes como robot_localization y nav2 son base; pero el reto agrícola es que el terreno se mueve, hay polvo, iluminación cambiante y vegetación deformable. Por eso, los modelos de incertidumbre y la reconfiguración dinámica de parámetros no son un lujo: son supervivencia.

La planeación global debe respetar mapas de costos (costmaps) que penalicen zonas blandas o pendientes; la planeación local tiene que amortiguar vibraciones y deslizamientos. Controladores como DWB o un MPC simplificado pueden funcionar si se recalibra para baja tracción. Una contribución formativa del proyecto es mostrar que la automatización agrícola no es “poner un LIDAR y ya”: es ingeniería de sistemas que cuadra percepción, planeación y actuadores (a veces hidráulicos) con bucles robustos.

El propósito del Meetup —“conectar industria con estudiantes, investigadores y entusiastas”— se materializa cuando un ponente internacional integra un simulador de ingeniería con ROS, y cuando grupos locales muestran humanoides, exápodos y robots agrícolas. Para la Distrital, esto traduce en un mapa de ruta pedagógico y de investigación:

1. Primer peldaño (formación básica)
   TurtleBots (o equivalentes) con ROS 2 Humble/Jazzy, rclpy, tópicos, servicios, TF2 y URDF. Laboratorios con Nav2 y ros2_control. Introducción a QoS y pruebas con archivos bag.

2. Segundo peldaño (integración y simulación)
   Ignition Gazebo, MoveIt (si hay brazos), robótica guiada por pruebas (TDD, gtests/pytest para nodos), CI con compilaciones reproducibles (colcon, vcs). Políticas de QoS explícitas por tópico, con tablas de latencias esperadas.

3. Tercer peldaño (aplicaciones especializadas)
   Slicer 2–ROS 2 para robótica médica, control de humanoides (estabilidad y MPC), robótica de campo (Nav2 extendido), y piernas (marchas y mapeo de elevación). A esto sumaría seguridad funcional (conceptos iniciales de ISO 13482/ISO 26262 adaptados) y telemetría/observabilidad (ros2_tracing, Prometheus/Grafana).

Ese cierre apunta a la ética del software libre: publicar, documentar, abrir incidencias (issues), y aceptar solicitudes de integración (pull requests). Para un ecosistema emergente como el colombiano, ese hábito vale tanto como cualquier artículo científico.

Lo más potente del Meetup es que reduce la distancia entre lo que vemos en clase y lo que exigen los robots reales:

- Diseñar interfaces ROS 2 antes del código. Definir tópicos, marcos TF, QoS y parámetros como contratos de ingeniería. Documentarlos en un archivo README y versionarlos.
- Medir latencias. No asumir que "publicar a 100 Hz" basta; perfilar duraciones de callbacks, utilizar ejecutores (executors) apropiados y, si es necesario, aislar nodos críticos en procesos separados.
- Simular con intención. No "jugar" en Gazebo; construir mundos que representen los riesgos (pendientes, colisiones blandas) y automatizar regresiones con pruebas de lanzamiento (launch tests).
- Validar en campo temprano. Así como el exápodo aprendió con baches, los robots de pasillo engañan.
- Apostar por herramientas puente. RosBlocks en semilleros y proyectos introductorios, Slicer en líneas biomédicas, y stacks de agricultura para prácticas en terreno con drones o UGVs.
- Participar en la comunidad. Más allá de consumir, contribuir: incidencias, documentación en español, bifurcaciones (forks) con mejoras mínimas pero útiles.

Si el capítulo logró reactivarse por activismo, nuestra contribución técnica puede ser la estandarización local: plantillas de colcon, lanzadores con perfiles de QoS, robot_localization preajustado a sensores disponibles en la Facultad, y ejemplos de ros2_control para los actuadores que realmente tenemos. Ese kit de arranque bajaría la fricción para nuevos equipos y haría que próximos Meetups muestren proyectos acumulativos, no islas. El ROSMeetup Bogotá 2025 no fue un catálogo de demostraciones vistosas; fue un recordatorio de que la ingeniería de robots es ingeniería de software con restricciones físicas. Tratar a Slicer como un nodo ROS 2 nos enseña a elegir interfaces limpias; controlar humanoides nos obliga a modelar y sintonizar; educar con RosBlocks muestra que la accesibilidad no está reñida con la rigurosidad; construir un exápodo en ROS 2 reafirma el valor de arquitecturas modulares; y navegar campos agrícolas nos pone frente al enemigo real: el mundo imperfecto.

Para la Universidad Distrital, que hoy alberga un capítulo activo y con mirada internacional, la tarea es sostener el impulso con prácticas repetibles, infraestructura compartida y publicaciones abiertas. Si logramos que nuestros robots "hablen ROS 2" con la misma disciplina con que un backend habla HTTP —contratos claros, tiempos de espera (timeouts), reintentos (retries), observabilidad—, entonces habremos dado el paso de comunidad activa a comunidad productiva.