NAV2

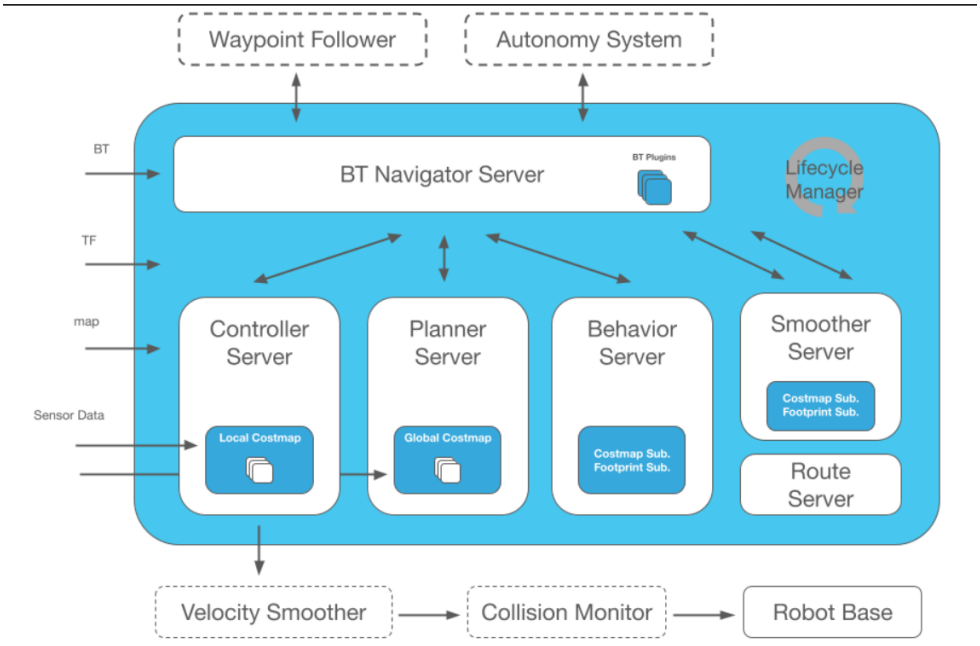
Overview:

Nav2 is a robust navigation framework trusted by over 50 global companies, offering a suite of features for building reliable autonomous systems.

1. **Open Navigation LLC**: This company provides leadership, maintenance, development, and support services to the Nav2 and ROS community[1].
2. **Dexory**: They are into developing robotics and AI logistics solutions to drive better business decisions using a digital twin of warehouses to provide inventory insights[12].
3. **Polymath Robotics**: Polymath Robotics creates safety-critical navigation systems for industrial vehicles that are radically simple to enable and deploy[1].
4. **Stereolabs**: This firm produces high-quality ZED stereo cameras with a complete vision pipeline from neural depth to SLAM, 3D object tracking, AI, and more[1].
5. **Crosswing Inc**: This company created Nav2 as a modular robotics platform, specifically an omniwheel holonomic mobile base with interchangeable upper components[3].

At its core, Nav2 employs behavior trees to orchestrate intelligent navigation behavior through various independent modular servers. These servers, tasked with different navigation-related tasks, communicate with the behavior tree over a ROS interface. Robots can utilize multiple behavior trees to perform a variety of unique tasks.

The inputs required by Nav2 include TF transformations, map sources, a BT XML file, and relevant sensor data. It then provides valid velocity commands for different types of robots, whether holonomic or non-holonomic, covering major robot types like differential-drive, legged, and car-like base types.



Some of the tools offered by Nav2 include:

- Map Server for handling maps.
- AMCL for robot localization on the map.
- Nav2 Planner for path planning.
- Nav2 Controller for robot control along the path.
- Nav2 Smoother for making path plans more continuous.
- Nav2 Costmap 2D for converting sensor data into a costmap representation.
- Nav2 Behavior Trees and BT Navigator for building complex robot behaviors.
- Nav2 Recoveries for computing recovery behaviors in case of failure.
- Nav2 Waypoint Follower for following sequential waypoints.
- Nav2 Lifecycle Manager for managing server lifecycles.
- Nav2 Core and other plugins for custom algorithms and behaviors.
- Collision Monitor for monitoring sensor data for potential collisions.
- Simple Commander for a Python3 API to interact with Nav2.
- Velocity Smoother for ensuring dynamic feasibility of velocity commands.

Navigation Concepts:

ROS2:

Action Server:

- Controls long-term tasks; communicates via NavigateToPose action message with Behavior Tree navigator; provides task feedback.
- Example: Requesting robot navigation, receiving time and distance feedback, obtaining success result.

Lifecycle Nodes:

- Unique to ROS 2; ensures deterministic system behavior during startup/shutdown.
- Example: Node transitions from unconfigured to active, processes information post-configuration, cleanly shuts down post-activation.

Behavior Trees:

Behavior Trees (BT) enhance the scalability and understandability of complex robotics tasks by organizing tasks in a tree structure, contrasting with the convoluted Finite State Machines (FSM). For instance, in a soccer-playing robot scenario, BT simplifies gameplay logic with basic primitives like "kick" or "go to ball." Utilizing the BehaviorTree CPP V3 library, this project constructs node plugins for navigation logic, ensuring verifiable, validated complex system creation. The library's subtree loading capability facilitates Nav2 behavior tree integration into higher-level BTs, like in soccer play BT. Additionally, the NavigateToPoseAction plugin enables

interaction with the Nav2 stack from client applications, hinting at potential future integration of Hierarchical FSMs (HFSM) based on user interest and contributions.

Navigation Servers:

Action Server:

Planners in Nav2 are tasked with computing paths to fulfill certain objectives, utilizing global environmental representations and buffered sensor data. They can be tailored to:

- Calculate shortest paths. (point A to B)
- Generate complete coverage paths.(cover an area)- maybe in mapping phase
- Devise paths along sparse or predefined routes.

Controller Server:

Controllers in Nav2, previously termed local planners in ROS 1, navigate globally computed paths or accomplish local tasks, utilizing local environmental representations to derive feasible control efforts. They can be designed to:

- Follow a path.
- Dock at charging stations using odometric frame detectors.
- Board elevators.
- Interface with tools.

Sends Velocity commands

Behaviors Server:

Recovery behaviors are crucial in fault-tolerant systems, addressing unknown or failure conditions autonomously. Examples include perception system faults leading to fake obstacles, or the robot being stuck due to dynamic obstacles or poor control, with actions like clearing costmaps or backing up to rectify these issues. In total failure cases, recoveries can alert operators via various communication channels. The behavior server hosts behaviors sharing access to resource-intensive elements like costmaps or TF buffers, each with a distinct API, not solely recovery behaviors.

Smoother Server:

Smoothers refine paths generated by planners, mitigating path raggedness and abrupt rotations, while also increasing distance from obstacles and high-cost areas, leveraging

global environmental representations. They offer the advantage of tailored smoothing, especially when combining different planners and smoothers or requiring specific path section smoothing. In Nav2, smoothers receive a path and return its improved version, accommodating various improvement criteria and methods for different input paths, thereby allowing a variety of smoothers to be registered in the server.

Waypoint follower:

Waypoint following in Nav2 enables navigation to multiple destinations via the `nav2_waypoint_follower` program, offering a plugin interface for task-specific executors. It demonstrates Nav2's utilization in applications ranging from simple demos to production-grade solutions, varying with the central dispatch's intelligence level. For smarter centralized dispatch, it suffices as an on-robot solution, while in other scenarios, it serves as a sample application, suggesting `nav2_behavior_tree` for creating custom, robust solutions. It also supports GPS waypoint following when coupled with `robot_localization`, converting GPS goals to Cartesian coordinates, facilitating diverse task management for robots in different environments.

State Estimation:

State Estimation in Nav2 requires two major transformations: map to odom, facilitated by a positioning system like localization or SLAM, and odom to base_link, provided by an odometry system. While there's no mandate for LIDAR usage in navigation, LIDAR-based solutions are supported and well-documented, although vision or depth-based positioning systems alongside other sensors for collision avoidance can also be utilized, provided they adhere to the community standards outlined in the project.

Standards:

REP 105 defines the frames and conventions required for navigation.

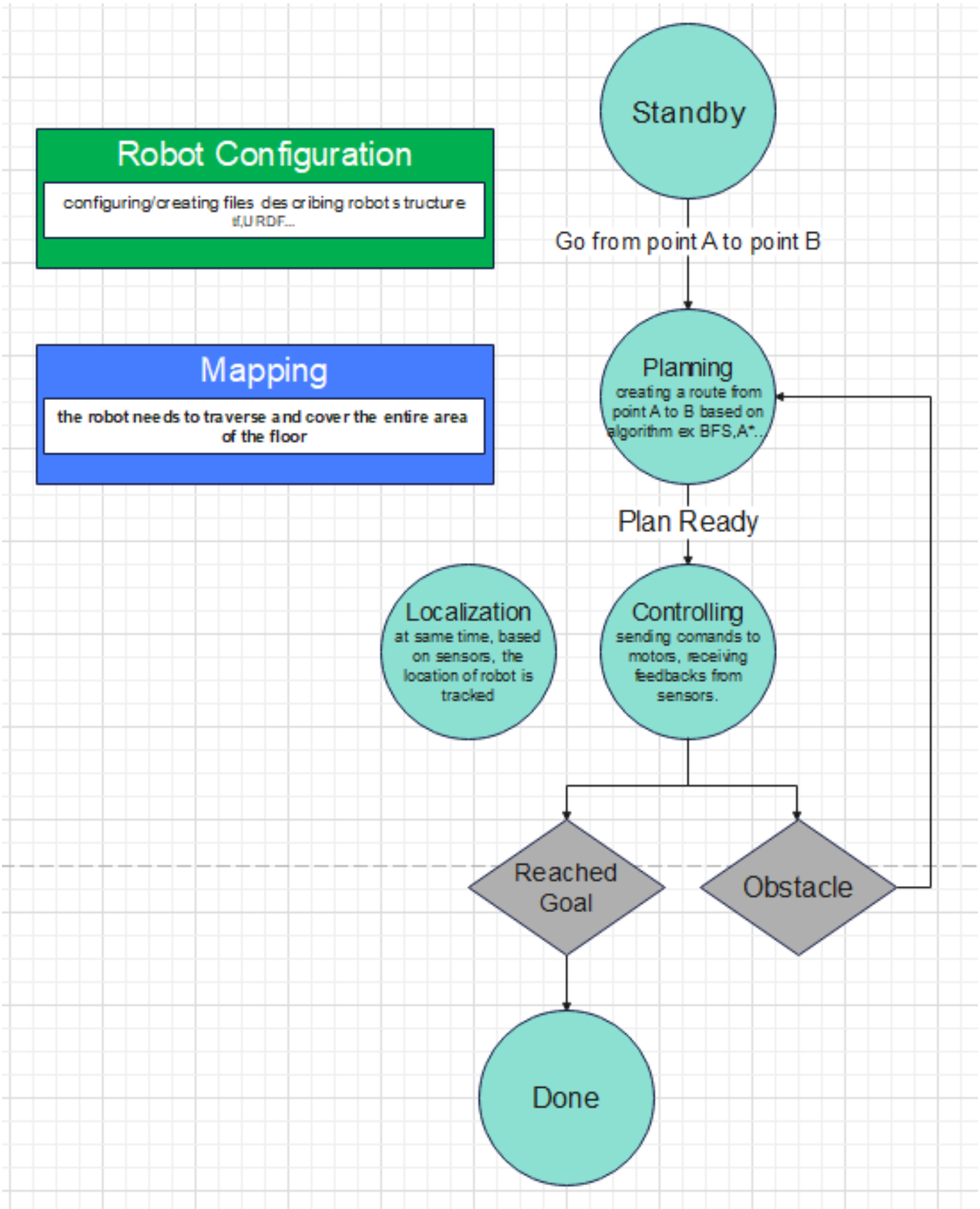
In a nutshell, REP-105 says that you must, at minimum, build a TF tree that contains a full map -> odom -> base_link -> [sensor frames] for your robot. TF2 is the time-variant transformation library in ROS 2 we use to represent and obtain time synchronized transformations. It is the job of the global positioning system (GPS, SLAM, Motion Capture) to, at minimum, provide the map -> odom transformation. It is then the role of the odometry system to provide the odom -> base_link transformation. The remainder of the transformations relative to base_link should be static and defined in your URDF.

Odometry:

It is the role of the odometry system to provide the odom -> base_link transformation. Odometry can come from many sources including LIDAR, RADAR, wheel encoders, VIO, and IMUs. The goal of the odometry is to provide a smooth and continuous local frame based on robot motion. The global positioning system will update the transformation relative to the global frame to account for the odometric drift.

Robot Localization is typically used for this fusion. It will take in N sensors of various types and provide a continuous and smooth odometry to TF and to a topic. A typical mobile robotics setup may have odometry from wheel encoders, IMUs, and vision fused in this manner.

The smooth output can be used then for dead-reckoning for precise motion and updating the position of the robot accurately between global position updates.


Robot Configuration
configuring/creating files describing robot structure
tf,URDF...
Mapping
the robot needs to traverse and cover the entire area of the floor
Standby
Go from point A to point B
Planning
creating a route from point A to B based on algorithm ex BFS,A*..
Plan Ready
Localization
at same time, based on sensors, the location of robot is tracked
Controlling
sending comands to motors, receiving feedbacks from sensors.
Reached Goal
Obstacle
Done

Robot Setup:

Step	Description
Setting Up Transformations	Understand/set up ROS transformations to translate/rotate coordinate frames.
Setting Up The URDF	Create a URDF file describing physical properties, dimensions, and joint dynamics.
Setting Up Odometry	Configure odometry for position and orientation data, essential for navigation.
Setting Up Sensors	Set up/calibrate sensors like RP LIDAR A1 for mapping and localization tasks.
Setting Up the Robot's Footprint	Define the robot's footprint for accurate collision avoidance during navigation.
Setting Up Navigation Plugins	Configure Nav2 plugins for path planning and control for autonomous navigation.