

Kết luận ngắn. Khung chương trình T800 đang nạp `move_base` qua interface `robot::move_base_core::BaseNavigation` và factory alias "`MoveBase`" có thể chạy `move_base2` mà không phải sửa host, nếu đối đúng thư viện và cây cấu hình runtime. Điều này đúng cho khung T800 hiện tại; không phải quy tắc tự động đúng cho mọi ứng dụng ROS dùng package `move_base`.

1. Khung đang khởi tạo những gì?

Log cho thấy chỉ có một node ROS là `/amr_node` cho phần điều khiển. Bên trong process đó, `amr_control` nạp động navigation runtime và các plugin. Global planner, local planner, action, recovery và mission adapter **không** là node ROS riêng.

`roslaunch` → `/amr_node (amr_control)` → `BaseNavigation factory` → `NavigationServer`

(`move_base2`)

→ `costmap global/local + runners + mission/action/recovery plugins`

Thành phần	Thời điểm / cách tạo	Vai trò quan sát được
<code>amr_control</code>	Node <code>/amr_node</code> ; tạo TF, localization, sensor converter, publisher/subscriber rồi nạp navigation.	Cầu nối ROS/MQTT/OPC-UA với lõi navigation.
<code>NavigationServer</code>	Factory của <code>libmove_base2.so</code> trả về <code>BaseNavigation::Ptr</code> .	Vô tương thích <code>BaseNavigation</code> ; quản lý runtime và control thread 30 Hz.
Global/local costmap	Tạo khi <code>NavigationRuntime::buildCostmaps()</code> chạy.	Global dùng frame <code>map</code> ; local dùng <code>odom</code> ; nhận laser, cloud, depth qua <code>SensorGateway</code> .
<code>PlannerRunner</code>	Nạp <code>CustomPlanner</code> lúc boot; nạp <code>SBPLLatticePlanner</code> , <code>DockPlanner</code> khi profile cần.	Lập global plan, cache instance theo tên planner.
<code>ControllerRunner</code>	Nạp <code>HybridLocalPlanner</code> lúc boot; có thể đổi theo profile.	Biến plan thành lệnh vận tốc.
<code>RecoveryRunner</code>	Nạp lúc boot từ namespace <code>recovery</code> .	Trong log: wait, clear-costmap (2 mức), rotate, back-up.
<code>ActionRunner</code>	Nạp lúc boot từ namespace <code>actions</code> .	Trong log: detect, wait, report,

sim_noop.

Mission layer

Nạp source adapter lúc boot từ
`mission_adapters`.

`GoalSourceAdapter` nhận goal;
`VDA5050SourceAdapter` tách order thành
các leg/nav/action.

Luồng một VDA5050 order

1. MQTT nhận topic order.
2. `VDA5050SourceAdapter` đổi order thành các mission leg: navigation hoặc action-only.
3. `MissionManager` đưa leg vào hàng đợi; `MissionExecutor` chạy tuần tự.
4. Control loop chọn profile: position, docking, go_straight hoặc rotate; runner lấy planner/local planner phù hợp.
5. Kết quả nav/action trả về mission layer, rồi trạng thái VDA5050.

2. C API đang dùng gì và có dùng được move_base2 không?

C API nằm tại `pnkx_nav_core/src/APIs/c_api`. Nó không tạo trực tiếp lớp C++ `move_base::MoveBase` hay `move_base2::NavigationServer`. Hàm `navigation_create()` làm đúng chuỗi sau:

```
PluginLoaderHelper::findLibraryPath("MoveBase")
boost::dll::import_alias<BaseNavigation::Ptr()>(path, "MoveBase")
factory() -> NavigationHandle
```

Vì `libmove_base2.so` export cả hai alias `MoveBase2` và `MoveBase`, C API sẽ nhận được một `NavigationServer` của `move_base2` khi cấu hình `MoveBase` trỏ đến `libmove_base2`. C API phía gọi không cần đổi tên hàm.

Nhóm C API	Ví dụ	Khi dùng move_base2
Vòng đời	<code>navigation_create</code> , <code>navigation_initialize</code> , <code>navigation_destroy</code>	Dùng được qua interface chung. Nên bảo đảm host gọi <code>shutdown()</code> ở đường C++ trước khi dỡ process/plugin.
Lệnh điều hướng	<code>navigation_move_to</code> , <code>navigation_move_to_order</code> , <code>navigation_dock_to</code> , <code>pause/resume/cancel</code>	Dùng được. Order/docking được <code>move_base2</code> đưa vào mission/profile tương ứng.
Sensor và map	<code>navigation_add_static_map</code> , <code>navigation_add_laser_scan</code> , <code>navigation_add_point_cloud2</code> , <code>odometry</code>	Dùng được; tên nguồn phải khớp source trong costmap YAML, ví dụ <code>pc_r_marking</code> .

Quan sát `navigation_get_feedback`, `pose/twist`, `global/local planner data` Dùng được cho trạng thái `BaseNavigation`. Không tự lộ API chi tiết của `MissionManager` hay `ActionRunner`.

Lưu ý kỹ thuật C API: đây là ABI C-linkage (hàm dùng `extern "C"`), nhưng header hiện có một số tham số tham chiếu C++ như `PoseStamped &out_pose` và `size_t &out_count`. Vì vậy nó **chưa** là header ISO C thuần để biên dịch trực tiếp bằng C compiler. Điều này không cản trở việc chọn `move_base2`, nhưng nếu caller là C thuần thì cần đổi các output reference thành pointer trước khi coi API là C API hoàn chỉnh.

3. `move_base2` khác `move_base` ở đâu?

Chủ đề	<code>move_base</code> cũ trong T800	<code>move_base2</code>
Ranh giới với host	<code>BaseNavigation</code> , plugin được nạp qua alias <code>MoveBase</code> .	Giữ cùng interface và xuất alias tương thích <code>MoveBase</code> ; thêm alias rõ ràng <code>MoveBase2</code> .
Tổ chức runtime	Điều phối kiểu đơn khối hơn.	Tách <code>NavigationRuntime</code> , <code>ControlLoop</code> , <code>PlannerRunner</code> , <code>ControllerRunner</code> , <code>RecoveryRunner</code> , <code>ActionRunner</code> và <code>MissionLayer</code> .
Nhiệm vụ / order	Host hoặc lớp ngoài thường phải tự điều phối nhiều bước.	Mission layer có adapter nguồn và executor; VDA5050 order được tách thành leg, action-only leg chạy qua <code>ActionRunner</code> .
Profile	Thường chỉ một bộ planner/controller cho một kiểu goal.	Profile position, docking, <code>go_straight</code> , <code>rotate</code> ; docking có marker profile. Planner/local planner có thể chuyển theo mission.
Fallback	Phụ thuộc implementation cũ.	Log chứng minh: khi <code>CustomPlanner</code> không hỗ trợ request đơn giản, runtime chuyển một lần sang <code>SBPLLatticePlanner</code> .
Safety	Tùy implementation.	Có điều kiện <code>require_current_costmap</code> ; sensor stale sẽ chặn wheel command để không điều khiển theo thể giới cũ.
Local planner plugin	Không nên giả định plugin cũ có cùng ABI.	Dùng contract <code>robot_nav_core2::LocalPlanner</code> ; local planner cũ chỉ dùng lại khi đã xác nhận cùng interface/ABI hoặc có adapter.

Đừng nhầm hai mức tương thích. Host/API `BaseNavigation` tương thích là một việc. Plugin local planner, YAML, và hành vi mission/profile tương thích là các việc khác. Việc đổi thư viện thành công không chứng minh tất cả planner cũ sẽ chạy được trong `move_base2`.

4. Một khung đang dùng move_base có chạy move_base2 luôn không?

Câu trả lời: có điều kiện. Với `amr_control` của T800 thì câu trả lời là **có**, theo đúng cơ chế đã được thiết kế. Với một khung ROS bất kỳ đang dùng package ROS1 `move_base`, câu trả lời là **không thể kết luận là có** nếu chưa kiểm tra interface.

Loại khung hiện có	Khả năng chuyển	Lý do / việc cần làm
T800 host nạp <code>BaseNavigation::Ptr</code> từ alias <code>MoveBase</code>	Cao	<code>move_base2</code> export alias tương thích. Đặt đúng library/config, rồi kiểm thử runtime.
Ứng dụng gọi C API <code>navigation_create()</code>	Cao	C API cũng tìm <code>MoveBase</code> ; config quyết định .so được nạp. Cần cùng ABI của <code>move_base_core</code> .
Ứng dụng liên kết trực tiếp class/private header của <code>move_base</code> cũ	Thấp	Phải sửa và build lại theo interface công khai hoặc làm adapter; không nên thay .so mù quáng.
ROS1 chuẩn dùng action <code>move_base_msgs/MoveBaseAction</code> và <code>pluginlib/nav_core</code>	Chưa khả định	Đây không phải tự động là contract T800 <code>BaseNavigation</code> ; cần kiểm tra node/action/topic/plugin ABI cụ thể.

Điều kiện bắt buộc để chuyển khung T800

- Chung ABI:** host, `libmove_base2.so` và `move_base_core` phải được build từ cùng workspace/devel hoặc ABI tương thích.
- Đúng alias:** library phải export factory `BaseNavigation::Ptr()` dưới tên `MoveBase`. `move_base2` hiện đã có alias này.
- Đúng config:** `PNKX_NAV_CORE_CONFIG_DIR` phải trỏ đến `move_base2/config/runtime`; file `move_base_common_params.yaml` cần có `MoveBase: library_path: libmove_base2`.
- Đủ plugin:** global planner/local planner/recovery/action/mission adapter được khai trong YAML phải có .so, alias factory và dependency đúng.
- Đúng sensor contract:** map, TF, odom, laser/cloud/depth được bơm bằng đúng topic-key và tần số. Với `require_current_costmap: true`, sensor stale sẽ chặn lệnh bánh xe.
- Shutdown rõ ràng:** dừng navigation trước khi destroy loader/process để tránh race thread/plugin.

Cách chuyển an toàn trong launch của T800

```
<!-- move_base2_control.launch đã làm hai việc quan trọng -->
<env name="PNKX_NAV_CORE_CONFIG_DIR"
      value="$(find move_base2)/config/runtime" />

# config/runtime/move_base_common_params.yaml
MoveBase:
  library_path: libmove_base2
```

Không đổi trực tiếp link library trong `amr_control`. Nó tiếp tục nạp factory "`MoveBase`"; file cấu hình chọn implementation là `move_base2`.

5. Checklist test trước khi thay cho hệ chạy dài

1. Build: xác nhận có `devel/lib/libmove_base2.so` và các plugin runtime.
2. Boot: log phải có `Found library ... libmove_base2.so, NavigationRuntime built`, costmap và sensor source tạo thành công.
3. Goal đơn: position, cancel/preempt, pause/resume; xác nhận `cmd_vel` và feedback.
4. Order: VDA5050 order có nhiều node/edge, action wait/detect/charge và docking marker.
5. Fallback/recovery: tạo tình huống CustomPlanner fail, obstacle/stale sensor có kiểm soát; xác nhận fallback/recovery và robot dừng an toàn.
6. Soak test: chạy 8 giờ, theo dõi CPU/RAM, tần số camera/cloud, message queue Gazebo, reconnect MQTT và tần suất sensor stale.

6. Liên hệ với cảnh báo qua đêm trong log

Cảnh báo `/gazebo/default/pose/local/info` là hàng đợi của Gazebo Transport đầy ở publisher đó; Gazebo bỏ một message để queue không tăng vô hạn. Nó không chứng minh C API hay `move_base2` bị lỗi. Tuy vậy, nó là dấu hiệu nên theo dõi tải mô phỏng/consumer. Cảnh báo ảnh hưởng an toàn hơn trong log là:

```
/camera_right/depth/points_proc observation buffer has not been updated ...  
[move_base2] Sensor data is stale – wheel commands blocked
```

`move_base2` đang dừng lệnh bánh xe đúng chủ đích vì local/global costmap không còn mô tả thế giới hiện tại. Cần chẩn đoán đường camera phải/DepthCameraData, callback SensorConverter và tần số thực tế; không quy kết ngay cho nghẽn mạng.

7. Dấu vết mã nguồn dùng để kết luận

- `Controllers/Packages/amr_control/src/amr_control.cpp`: host tìm library `MoveBase` và import factory alias.
- `Test/move_base2/src/move_base2_plugin.cpp`: factory trả `BaseNavigation::Ptr`, export cả `MoveBase2` và `MoveBase`.
- `Test/move_base2/launch/move_base2_control.launch`: đặt `PNKX_NAV_CORE_CONFIG_DIR` sang overlay runtime.
- `Test/move_base2/config/runtime/move_base_common_params.yaml`: `MoveBase.library_path = libmove_base2`.
- `pnkx_nav_core/src/APIs/c_api/src/nav_c_api.cpp`: `navigation_create()` dùng chính alias `MoveBase`.
- `pnkx_nav_core/src/Navigations/Cores/move_base_core/include/move_base_core/navigation.h`: contract chung `BaseNavigation`.

- `Test/move_base2/src/navigation_runtime.cpp`: xây costmap, runners và mission layer.