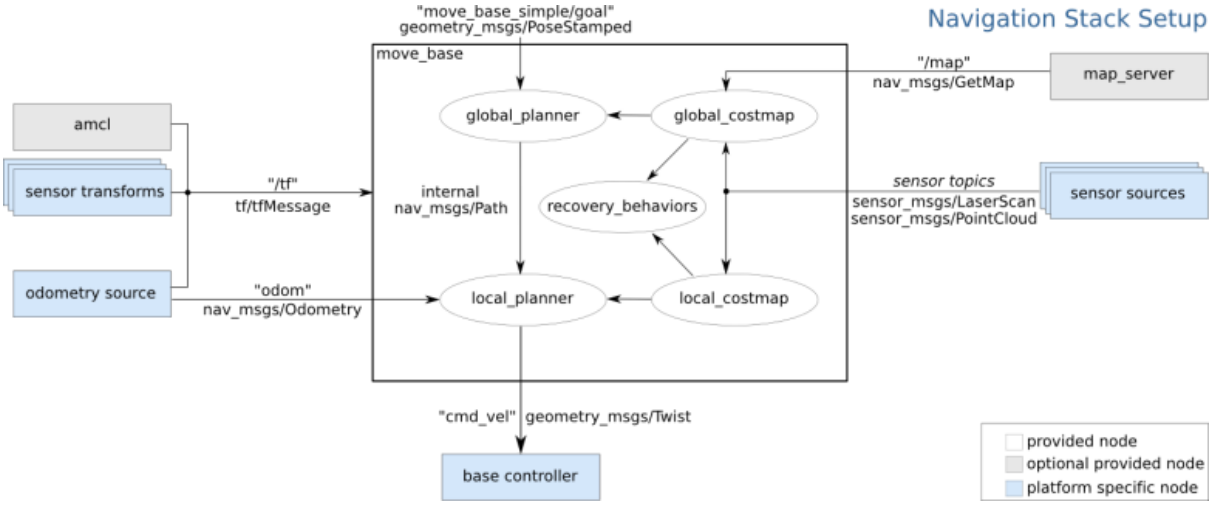
بسمه تعالی

پشته ناوبری یک بسته آماده است که برای ربات به نحوی خاص به منظور اجرا پیکربندی شده است.



نمودار بالا یک نمای کلی از این پیکربندی را نشان می دهد.
۱-اجزای سفید مورد نیاز اجزای مورد نیاز هستند،
۲- اجزای خاکستری اجزای اختیاری هستند که قبلا اجرا شده اند
۳- اجزای آبی برای هر پلت فرم ربات باید ایجاد شوند.

پیش نیازهای پشته ناوبری همراه با دستورالعمل هایی درباره نحوه برآورده ساختن هر مورد، در بخش های زیر ارائه شده است.

ROS
پشته ناوبری فرض می کند که ربات از ROS استفاده می کند. لطفا از مستندات ROS برای راهنمایی در مورد چگونگی نصب ROS روی ربات خود، مشورت بگیرید.

پیکربندی تبدیل (تبدیل دیگر)

پیکربندی تبدیل با استفاده از بخشی از ربات انجام میشود که اطلاعات مربوط به روابط بین فریم های مختصات را با استفاده از TF انتشار می دهد. آموزش مفصل در راه اندازی این پیکربندی در اینجا می توانید پیدا کنید.

http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF

خلاصه TF

پیکربندی تبدیل

بسیاری از بسته های ROS نیاز به تبدیل درخت ربات را با استفاده از کتابخانه نرم افزاری TF منتشر می کنند. در یک سطح انتزاعی، یک درخت تبدیل تعریف ها را از لحاظ ترجمه و چرخش بین فریم های مختصات مختلف تعریف می کند.

برای ساخت این Base بیشتر، نمونه ای از یک ربات ساده که پایه تلفن همراه را با یک لیزر تک در بالای آن نصب کرده است را در نظر بگیرید. در اشاره به ربات، دو فریم مختصات را تعریف می کنیم: یکی مربوط به نقطه مرکزی پایه ربات و یکی برای نقطه مرکزی لیزر است که در بالای پایه نصب شده است. بیایید نام آنها را برای مرجع آسان نیز ارائه دهیم. ما کادر مختصات متصل به پایه تلفن همراه base_link (برای ناوبری، مهم است که این در مرکز چرخشی ربات قرار می گیرد) تماس می گیریم و ما کادر هماهنگی متصل به لیزر base_laser را فراخوانی می کنیم. برای قراردادهای نامگذاری قاب، REP 105 را ببینید.

در این مرحله، فرض کنیم که ما داده هایی از لیزر را به صورت فاصله ای از نقطه مرکز لیزر داریم. به عبارت دیگر، ما بعضی از داده ها را در قاب هماهنگ "base_laser" داریم. اکنون فرض کنید ما می خواهیم این داده ها را بگیریم و از آن برای کمک به تلفن همراه استفاده کنیم .
تا از برخورد با موانع در جهان جلوگیری کنیم. برای انجام این کار که با موفقیت صورت گیرد ، ما نیاز به یک روش برای تبدیل اسکن لیزری که از کادر frame__laser گرفته شده به فریم base_link گرفته ایم. در اصل، ما باید یک رابطه بین فریم مختصات "base_laser" و "base_link" تعریف کنیم.

اطلاعات سنسور (منابع حسگر)
پشته ناوبری از اطلاعاتی از سنسورها برای جلوگیری از موانع در جهان استفاده می کند، فرض می کند که این سنسورها پیام های
sensor_msgs / LaserScan یا sensor_msgs / PointCloud را از طریق ROS انتشار می دهند. برای کسب اطالعات در مورد انتشار این پیام ها از طریق ROS، لطفا به بررسی (جریان های انتشار) ،(انتشار سنسور) ROS مراجعه کنید.

نکته:
درایور یا راه انداز که Driver گفته میشود برای شناساندن یک قطعه سخت افزاری به سیستم عامل ربات استفاده میشود .
همچنین، تعدادی از سنسورها دارای درایورROS هستند که قبلا از این مرحله عمل مراقبت را انجام میدهند . سنسورهای پشتیبانی شده و پیوندهای مربوط به درایور مناسب آنها در زیر ذکر شده است:

دستگاه لیزر Hokuyo سازگار با SCIP2.2 و همچنین مدل Hokuyo 04LX، 30LX - urg_node
SICK LMS2xx Lasers - sicktoolbox_wrapper

http://wiki.ros.org/urg_node
http://wiki.ros.org/sicktoolbox_wrapper

اطلاعات سنجش از دور (هندسی منبع)

پشته ناوبر نیازمند این است که اطلاعات سنجش با استفاده از TF و پیام نوگرا / پیام Odometry منتشر شود.

آموزش در مورد انتشار اطلاعات Odometry را در اینجا می توانید پیدا کنید:

http://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom

انتشار اطلاعات Odometry بیش از ROS. سیستم عامل های پشتیبانی شده برای Odometry و پیوندهایی با درایورهای مناسب آنها در زیر آورده شده است:

Videre Erratic: erratic_player
PR2: pr2_mechanism_controllers

کنترل کننده پایه (کنترل پایه)
پشته ناوبری فرض می کند که می تواند دستورات سرعت را با استفاده از یک پیام geometry_msgs / Twist فرض می کند که در کادر هماهنگ پایه ربات در موضوع cmd_vel قرار دارد.
این به این معنی است که باید یک گره مشترک به موضوع cmd_vel باشد که بتواند از آن استفاده کند ((vx، vy، vtheta velocity (cmd_vel.linear.x، cmd_vel.linear.y، cmd_vel.angular.z) <==> و تبدیل آنها به فرمان های موتور برای ارسال به یک پایگاه همراه. سیستم عامل های پشتیبانی شده برای کنترل پایه و لینک ها به رانندگان مناسب آنها در زیر ذکر شده است:

Videre خراب: erratic_player
PR2: pr2_mechanism_controllers

نقشه برداری (map_server)
پشته ناوبر نیازمند به یک نقشه برای کار نیست، اما برای اهداف این آموزش، فرض می کنیم که شما یکی را داشته باشید. برای اطلاعات بیشتر در مورد ایجاد یک نقشه از محیط زیست، لطفا یک ساختمان نقشه را ببینید.

http://wiki.ros.org/slam_gmapping/Tutorials/MappingFromLoggedData

تنظیم پیکربندی ناوبری

این بخش نحوه راه اندازی و پیکربندی پشته ناوبری را در یک ربات توضیح می دهد. با فرض این که تمام الزامات فوق برای تنظیم روبات مورد تایید باشد.

به طور خاص، اربات باید اطلاعات فریم مختصات را با استفاده از tf ارسال کند، پیام sensor_msgs / LaserScan یا sensor_msgs / PointCloud را از همه سنسورهایی که باید با ستون ناوبری استفاده شوند، و انتشار اطلاعات Odometry با استفاده از هر دو tf و nav_msgs / Odometry پیام در حالی که همچنین در دستورات سرعت برای ارسال به پایگاه. اگر هر یک از این الزامات در ربات شما برآورده نشود،

لطفا دستورالعمل مربوط به تکمیل آنها را در بخش تنظیمات ربات بالا مشاهده کنید.

http://wiki.ros.org/slam_gmapping/Tutorials/MappingFromLoggedData

ایجاد یک بسته

این اولین قدم برای این آموزش ایجاد بسته ای است که در آن تمام پیکربندی و راه اندازی فایل ها برای پشته ناوبری ذخیره می شود.

این بسته وابسته به هر بسته ای است که مورد نیاز برای اجرای الزامات در بخش تنظیم ربات در بالا و همچنین در بسته move_base که شامل رابط سطح بالا به پشته ناوبری است، وابسته است. بنابراین، یک مکان را برای بسته خود انتخاب کنید و دستور زیر را اجرا کنید:

```
catkin_create_pkg my_robot_name_2dnav move_base my_tf_configuration_dep
my_odom_configuration_dep my_sensor_configuration_dep
```

این دستور یک بسته با وابستگی های لازم برای اجرای پشته ناوبری روی ربات شما ایجاد می کند.

ایجاد پرونده راه اندازی پیکربندی ربات

حالا که ما یک فضای کاری برای تمام پیکربندی و راه اندازی فایل های ما داریم، یک فایل rollaunch ایجاد می کنیم که تمام سخت افزار را به ارمغان می آورد و انتشار هایی را که ربات نیاز دارد تبدیل می کند. ویرایشگر مورد علاقه خود را بکشید و قطعه بعدی را به یک فایل به نام my_robot_configuration.launch بزنید. البته، شما باید، البته، متن «my_robot» را با نام ربات واقعی خود جایگزین کنید. ما همچنین باید تغییرات مشابهی را در فایل راه اندازی به شرح زیر انجام دهیم، پس مطمئن شوید که بقیه این بخش را بخوانید.

```
odom_node_pkg" type="odom_node_type" name="odom_node" output="screen">
    <param name="odom_param" value="param_value" />
 </node>
 <node pkg="transform_configuration_pkg" type="transform_configuration_type"
name="transform_configuration_name" output="screen">
    <param name="transform_configuration_param" value="param_value" />
 </node>
```

```
</launch>
```

خوب .. حالا ما یک قالب برای یک فایل راه اندازی داریم، اما باید آن را برای ربات خاصی پر کنیم. ما از طریق تغییراتی که باید در هر بخش زیر ایجاد کنیم، راه می رود.

```
<launch>

  <node pkg="sensor_node_pkg" type="sensor_node_type" name="sensor_node_name"
output="screen">
```

جای "sensor_node_pkg" را با نام بسته برای "ROS Drive " برای "حسگر خود، "sensor_node_type" با " Type Drives" برای "سنسور خود" جایگزین کنید "sensor_node_name" با نام مورد نظر برای "sensor node شما" و "sensor_param" با هر پارامتری که "Node شما" امکان آن است.

توجه داشته باشید که اگر شما چندین سنسور که قصد استفاده از آنها را برای ارسال اطلاعات به پشته ناوبری داشته باشید، باید همه آنها را در اینجا راه اندازی کنید.

```
</node>
 <node pkg="odom_node_pkg" type="odom_node_type" name="odom_node"
output="screen">
   <param name="odom_param" value="param_value" />
```

در این بخش، Odometry را برای پایه راه اندازی خواهیم کرد. یک بار دیگر، شما باید مشخصات pkg، نوع، نام و پارامترآن را به آنهایی که مربوط به گره ای هستند که راه اندازی کرده اید را جایگزین می کنید.

```
  <param name="transform_configuration_param" value="param_value" />
 </node>
```

نوع، نام و ،pkg در این بخش، ما تنظیمات تبدیل برای ربات را راه اندازی خواهیم کرد. یک بار دیگر، شما باید مشخصات b.پارام را با آنهایی که مربوط به گره ای هستند که در واقع راه اندازی می کنید را جایگزین کنید

پیکربندی هزینه (local_costmap) و (global_costmap)
پشته ناوبری با استفاده از دو costmaps برای ذخیره اطلاعات در مورد موانع در جهان است. یکی از طرح های هزینه برای برنامه ریزی جهانی استفاده می شود، بدین معنی که ایجاد برنامه های بلند مدت در کل محیط زیست، و دیگری برای برنامه ریزی محلی و جلوگیری از مانع استفاده می شود. برخی از گزینه های پیکربندی وجود دارد که ما می خواهیم هر دو costmaps به دنبال داشته باشند، و برخی که ما می خواهیم روی هر نقشه به صورت جداگانه تنظیم کنیم. بنابراین، برای پیکربندی costmap سه گزینه زیر وجود دارد: گزینه های پیکربندی رایج، گزینه های پیکربندی جهانی و گزینه های پیکربندی محلی.

گزینه ها، لطفا مستندات costmap_2d را ببینید

پیکربندی معمول (local_costmap) و (global_costmap)

پشته ناوبری از costmaps برای ذخیره اطلاعات در مورد موانع در جهان استفاده می کند. برای انجام این کار به درستی، ما باید نقاط هزینه در موضوعات حسگر که باید برای به روز رسانی گوش دادن به آنها اشاره کنیم، اشاره کنیم. یک فایل با نام costmap_common_params.yaml ایجاد کنید، همانطور که در زیر نشان داده شده است و آن را پر کنید:

```
obstacle_range: 2.5
raytrace_range: 3.0
footprint: [[x0, y0], [x1, y1], ... [xn, yn]]
#robot_radius: ir_of_robot
inflation_radius: 0.55

observation_sources: laser_scan_sensor point_cloud_sensor

laser_scan_sensor: {sensor_frame: frame_name, data_type: LaserScan, topic: topic_name,
marking: true, clearing: true}

point_cloud_sensor: {sensor_frame: frame_name, data_type: PointCloud, topic: topic_name,
marking: true, clearing: true}
```

خوب .. حالا ما یک قالب برای یک فایل راه اندازی داریم، اما باید آن را برای ربات خاصی پر کنیم. ما از طریق تغییراتی که باید در هر بخش زیر ایجاد کنیم، راه می رود.

```
 <node pkg="sensor_node_pkg" type="sensor_node_type" name="sensor_node_name"
output="screen">
```

در این بخش، ما هر سنسورهایی را که ربات برای ناوبری استفاده می کند آورده است. replace sensor_node_pkg با نام بسته برای راننده ROS برای سنسور خود، sensor_node_type را با نوع راننده برای سنسور خود، " sensor_node_name" با نام دلخواه برای گره حسگر خود و "sensor_param" با هر پارامتر که گره شما ممکن است. توجه داشته باشید که اگر شما چندین سنسور که قصد استفاده از آنها را برای ارسال اطلاعات به پشته ناوبری داشته باشید، باید همه آنها را در اینجا راه اندازی کنید.

```
</node>
 <node pkg="odom_node_pkg" type="odom_node_type" name="odom_node"
output="screen">
   <param name="odom_param" value="param_value" />
 </node>
```

در این بخش، Odometry را برای پایه راه اندازی خواهیم کرد. یک بار دیگر، شما باید مشخصات pkg، نوع، نام و پارام را با آنهایی که مربوط به گره ای هستند که در واقع راه اندازی می کنید را جایگزین کنید.

<param name="transform_configuration_param" value="param_value" />
</node>

در این بخش، ما تنظیمات تبدیل برای ربات را راه اندازی خواهیم کرد. یک بار دیگر، شما باید مشخصات pkg، نوع، نام و پارام را با آنهایی که مربوط به گره ای هستند که در واقع راه اندازی می کنید را جایگزین کنید.

پیکربندی هزینه (local_costmap) و (global_costmap)
پشته ناوبری با استفاده از دو costmaps برای ذخیره اطلاعات در مورد موانع در جهان است. یکی از طرح های هزینه برای برنامه ریزی جهانی استفاده می شود، بدین معنی که ایجاد برنامه های بلند مدت در کل محیط زیست، و دیگری برای برنامه ریزی محلی و جلوگیری از مانع استفاده می شود. برخی از گزینه های پیکربندی وجود دارد که ما می خواهیم هر دو costmaps به دنبال داشته باشند، و برخی که ما می خواهیم روی هر نقشه به صورت جداگانه تنظیم کنیم. بنابراین، برای پیکربندی costmap سه گزینه زیر وجود دارد: گزینه های پیکربندی رایج، گزینه های پیکربندی جهانی و گزینه های پیکربندی محلی.

نکته: بخش های زیر تنها گزینه های پیکربندی اولیه برای هزینه را پوشش می دهد. برای مستند سازی در طیف وسیعی از گزینه ها، لطفا مستندات costmap_2d را ببینید.

پیکربندی معمول (local_costmap) و (global_costmap)

پشته ناوبری از costmaps برای ذخیره اطلاعات در مورد موانع در جهان استفاده می کند. برای انجام این کار به درستی، ما باید نقاط هزینه در موضوعات حسگر که باید برای به روز رسانی گوش دادن به آنها اشاره کنیم، اشاره کنیم. یک فایل با نام costmap_common_params.yaml ایجاد کنید، همانطور که در زیر نشان داده شده است و آن را پر کنید:

obstacle_range: 2.5
raytrace_range: 3.0
footprint: [[x0, y0], [x1, y1], ... [xn, yn]]
#robot_radius: ir_of_robot
inflation_radius: 0.55

observation_sources: laser_scan_sensor point_cloud_sensor

laser_scan_sensor: {sensor_frame: frame_name, data_type: LaserScan, topic: topic_name, marking: true, clearing: true}

point_cloud_sensor: {sensor_frame: frame_name, data_type: PointCloud, topic: topic_name, marking: true, clearing: true}

خوب، بیایید فایل بالا را به قسمت های مدیریتی برسانیم.

obstacle_range: 2.5
raytrace_range: 3.0

این پارامترها آستانه ها را بر روی اطلاعات مانع قرار داده شده در صفحه هزینه می کند. پارامتر "barrier_range" حداکثر خواندن سنسور محدوده را تعیین می کند که باعث می شود مانع در costmap شود. در اینجا، ما آن را در 2.5 متر تنظیم شده است، که به این معنی است که ربات فقط نقشه خود را با اطلاعات در مورد موانعی که در فاصله 2.5 متر از پایه قرار دارند به روز می کند. پارامتر "raytrace_range" دامنه ای را تعیین می کند که ما با استفاده از یک خواندن سنسور فضای آزاد raytrace را می بینیم. تنظیم آن به 3.0 متر همانطور که در بالا اشاره کردیم به این معنی است که ربات سعی خواهد کرد فاصله آن را از مقابل فاصله 3.0 متر با توجه به خواندن حسگر پاک کند.

footprint: [[x0, y0], [x1, y1], ... [xn, yn]]
#robot_radius: ir_of_robot
inflation_radius: 0.55

در اینجا ما رد پای ربات یا شعاع ربات را اگر دایره ای است تعیین کنیم. در مورد تعیین ردیابی، مرکز ربات در نظر گرفته شده است (0.0، 0.0) و هر دو در جهت عقربه های ساعت و مشخصات ضد clockwiz پشتیبانی می شوند. ما همچنین شعاع تورم را برای قیمت میگیریم. شعاع تورم باید به حداکثر فاصله تا موانع که هزینه آن باید متحمل شود تنظیم شود. به عنوان مثال، تنظیم شعاع تورم در 0.55 متر به این معنی است که ربات تمام مسیرهایی را که 0.55 متر یا بیشتر از موانع باقی می مانند، به عنوان هزینه برخورد به مانع برابر در نظر میگیرد.

observation_sources: laser_scan_sensor point_cloud_sensor

پارامتر "observation_sources" لیستی از سنسورها را تنظیم می کند که اطلاعات را به فرایند جداگانه از فضاهای جداگانه منتقل می کنند. هر سنسور در خطوط بعدی تعریف شده است.

laser_scan_sensor: {sensor_frame: frame_name, data_type: LaserScan, topic: topic_name, marking: true, clearing: true}

این خط پارامترها را بر روی حسگر ذکر شده در مشاهدات__ منابع تعیین می کند، و در این مثال، laser_scan_sensor را به عنوان مثال تعریف می کند. پارامتر "frame_name" باید بر اساس نام کادر هماهنگ سنسور تنظیم شود، پارامتر data_type باید به LaserScan یا PointCloud بر اساس کدام پیام مورد استفاده قرار گیرد و نام topic_name باید بر

روی نام قرار گیرد از موضوعی که سنسور آن را منتشر می کند. پارامترهای "علامت گذاری" و "پاکسازی" تعیین می کند که آیا از این حسگر برای اضافه کردن اطلاعات مانع به قیمت به کار می رود، اطلاعات مانع را از طریق costmap روشن می کند یا هر دو را انجام می دهد.

پیکربندی جهانی (global_costmap)

ما یک فایل زیر ایجاد خواهیم کرد که گزینه های پیکربندی خاص مربوط به هزینه های جهانی را ذخیره می کند. یک ویرایشگر با فایل global_costmap_params.yaml را باز کنید و در متن زیر قرار دهید:

```
global_costmap:
  global_frame: /map
  robot_base_frame: base_link
  update_frequency: 5.0
  static_map: true
```

پارامتر "global_frame" تعریف می کند که چه چارچوب مختصات، باید costmap اجرا شود، در این صورت، ما frame / map را انتخاب می کنیم. پارامتر "robot_base_frame" فریم مختصات را تعریف می کند که costmap باید برای پایه ربات مرجع باشد. پارامتر "update_frequency" فرکانس را در هرتز تعیین می کند که در آن costmap حلقه به روز رسانی خود را اجرا می کند. پارامتر "static_map" تعیین می کند که آیا باید هزینه اولیه خود را براساس یک نقشه ارائه شده توسط map_server آغاز دهیم یا نه. اگر از یک نقشه نقشه یا نقشه موجود استفاده نمی کنید، پارامتر static_map را به false تنظیم کنید.

پیکربندی محلی (local_costmap)

ما یک فایل زیر ایجاد می کنیم که گزینه های پیکربندی خاصی را برای هزینه های محلی ذخیره می کند. یک ویرایشگر با فایل local_costmap_params.yaml را باز کنید و در متن زیر قرار دهید:

```
local_costmap:
  global_frame: odom
  robot_base_frame: base_link
  update_frequency: 5.0
  publish_frequency: 2.0
  static_map: false
  rolling_window: true
  width: 6.0
  height: 6.0
  resolution: 0.05
```

پارامترهای "update_frequency" ،"robot_base_frame" ،"global_frame" و "static_map" همانند در بخش تنظیمات جهانی در بالا شرح داده شده است. پارامتر "publish_frequency" نرخ، در هرتز تعیین می کند که در آن قیمت هزینه اطلاعات تجسم را منتشر می کند. تنظیم پارامتر "null_window" به درست است بدین معناست که قیمت در اطراف ربات باقی خواهد ماند در حالی که ربات از طریق جهان حرکت می کند. پارامترهای "عرض"، "ارتفاع" و "رزولوشن" عرض پیکره (متر)، ارتفاع (متر) و رزولوشن (متر / سلول) از هزینه را تنظیم می کند. توجه داشته باشید که جریمه نقدی برای حل این شبکه متفاوت از تفکیک نقشه ایستا است، اما اغلب اوقات ما تمایل داریم آنها را به طور هماهنگ تنظیم کنیم.

گزینه های پیکربندی کامل

این حداقل پیکربندی باید کارها را انجام داده و اجرا کند، اما برای جزئیات بیشتر در مورد گزینه های پیکربندی موجود برای هزینه، لطفا مستندات costmap_2d را ببینید.

پیکربندی Planner محلی پایه
base_local_planner مسئول محاسبه دستورات سرعت برای ارسال به پایانه تلفن همراه ربات با توجه به یک سطح بالای سطح است. ما نیاز به تنظیم برخی از گزینه های پیکربندی بر اساس مشخصات ربات ما برای به دست آوردن همه چیز و در حال اجرا است. یک فایل با نام base_local_planner_params.yaml باز کنید و متن زیر را در آن قرار دهید:

توجه: این قسمت تنها گزینه های پیکربندی اولیه برای TrajectoryPlanner را پوشش می دهد. برای مستند سازی در طیف وسیعی از گزینه ها، لطفا مستندات base_local_planner را ببینید.

```
TrajectoryPlannerROS:
  max_vel_x: 0.45
  min_vel_x: 0.1
  max_vel_theta: 1.0
  min_in_place_vel_theta: 0.4

  acc_lim_theta: 3.2
  acc_lim_x: 2.5
  acc_lim_y: 2.5

  holonomic_robot: true
```

بخش اول پارامترهای فوق محدودیت سرعت ربات را تعیین می کند. بخش دوم، محدودیت شتاب ربات را تعیین می کند.

ایجاد یک فایل راه اندازی برای Stack Navigation
حالا که تمام پرونده های پیکربندی ما را در جای خود قرار داده ایم، باید همه چیز را به یک فایل راه اندازی برای پشته ناوبری برسانیم. یک ویرایشگر را با فایل move_base.launch باز کنید و متن زیر را در آن بگذارید:

```
launch>

  <master auto="start"/>
<!-- Run the map server -->
  <node name="map_server" pkg="map_server" type="map_server" args="$(find
```

```
my_map_package)/my_map.pgm my_map_resolution"/>

 <!--- Run AMCL -->
   <include file="$(find amcl)/examples/amcl_omni.launch" />

  <node pkg="move_base" type="move_base" respawn="false" name="move_base"
output="screen">
   <rosparam file="$(find my_robot_name_2dnav)/costmap_common_params.yaml"
command="load" ns="global_costmap" />
   <rosparam file="$(find my_robot_name_2dnav)/costmap_common_params.yaml"
command="load" ns="local_costmap" />
   <rosparam file="$(find my_robot_name_2dnav)/local_costmap_params.yaml"
command="load" />
   <rosparam file="$(find my_robot_name_2dnav)/global_costmap_params.yaml"
command="load" />
   <rosparam file="$(find my_robot_name_2dnav)/base_local_planner_params.yaml"
command="load" />
 </node>

</launch>
```

تنها تغییراتی که باید برای ایجاد این فایل انجام دهید این است که سرور نقشه را تغییر دهید تا به نقشه ای که ایجاد کرده اید اشاره کنید و اگر شما یک ربات دیفرانسیل دیجیتال دارید، amcl_omni.launch را به amcl_diff.launch تغییر دهید. برای یک آموزش در مورد ایجاد یک نقشه، لطفا ساختمان نقشه را ببینید.

پیکربندی (AMCL) AMCL
AMCL گزینه های پیکربندی زیادی دارد که بر عملکرد محلی سازی تاثیر می گذارد. برای اطلاعات بیشتر در مورد AMCL لطفا به اسناد amcl مراجعه کنید.

در حال راه اندازی پشته حرکت
حالا ما همه چیز را تنظیم کرده ایم، ما می توانیم پشته ناوبری را اجرا کنیم. برای انجام این کار، ما به دو پایانه در ربات نیاز داریم. در یک ترمینال، ما فایل my_robot_configuration.launch را راه اندازی می کنیم و از سوی دیگر فایل file move_base.launch را که ما آن را ایجاد کرده ایم راه اندازی می کنیم.

```
roslaunch my_robot_configuration.launch
```

```
roslaunch move_base.launch
```

تبریک می گویم، پشته ناوبری باید در حال اجرا باشد. برای اطلاعات در مورد ارسال اهداف به پشته ناوبری از طریق یک رابط گرافیکی، لطفا از آموزش rviz و ناوبری دیدن کنید. اگر میخواهید به جای استفاده از کد، از اهداف خود به ستون ناوبری ارسال کنید، لطفا به آموزش ساده ارسال هدایت ناوبری مراجعه کنید.

عیب یابی
برای مسائل رایج هنگام اجرای پشته ناوبری، لطفا صفحه عیب یابی پشته ناوبری را مشاهده کنید.

#keywords راه اندازی پلت فرم تلفن همراه، راه اندازی ربات، ربات راه اندازی، شروع به کار با ربات موبایل