قسمت اول)

در این قسمت ابتدا نسخه‌ی ROS مورد نظر بر روی سیستم عامل لینوکس نصب گردید و بعد از آن مراحل گفته شده در مقاله‌ی داده‌شده بررسی و اعمال شد که در آن ابتدا با ساختار workspaceهای ROS آشنایی مختصری صورت گرفت و پس از آن پکیجی با نام random_control ساخته شد که درون آن سه گره به اسامی random_values برای تولید مقادیر تصادفی، move_robot برای دستورات حرکتی ربات، pose_monitor برای مانیتور کردن وضعیت ربات در نظر گرفته شد. با این تفاسیر پس از اجرای شبیه‌ساز و اجرای این گره‌ها، ربات مورد نظر به صورت تصادفی در محیط حرکت خواهد کرد که در زیر دستوراتی که برای اجرای آن مدنظر است، آورده شده است:

```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
rosrun random_control random_values.py
rosrun random_control move_robot.py
rosrun random_control pose_monitor.py
```

قسمت دوم)

مقدمه

در این قسمت به سراغ قسمت اصلی تمرین می‌رویم که باید با ساخت یک پکیج و دو گره، حرکت ربات ما به صورتی باشد که یک مربع به مرکز مبدا مختصات و به ضلع ۳ متر را پیمایش کند.

نکات مدل‌سازی و پیاده‌سازی

برای این منظور یک پکیج با نام square_control می‌سازیم و درون آن دو گره که اولی pose_monitor است و برای نمایش موقعیت ربات در لحظات مختلف، محاسبه خطا و کمی‌سازی آن و همچنین نمایش مسیر طی شده توسط ربات به صورت گرافیکی و گره‌ی دومی move_robot نام دارد که وظیفه‌ی آن به این صورت است که رباتی با سرعت خطی ثابت ۰.۹ متر بر ثانیه تنظیم شده‌است و روی مسیر که منظور همان مربع است، ۲۴ نقطه مشخص شده است و تلاش می‌شود که ربات در هر لحظه به سمت بهترین هدفی که پیش رو دارد حرکت کند و زاویه فرمان ربات با توجه به این موارد تغییر کند.

در راستای تغییر زاویه فرمان چندین ثابت در نظر گرفته شده؛ به عنوان مثال همانند مطالب موجود در درس، یک ثابت k_theta در نظر گرفته شده که به یک‌باره فرمان با سرعت نپیچد و ربات سُر نخورد. ثابت دیگری که در نظر گرفتیم، ثابت d بوده به این معنی که هنگامی که به فاصله‌ی d از یک هدف رسیدیم، هدف ربات را نقطه‌ی بعدی در مسیر قرار دهیم تا زودتر از رسیدن به مقصد، فرمان بچرخد.

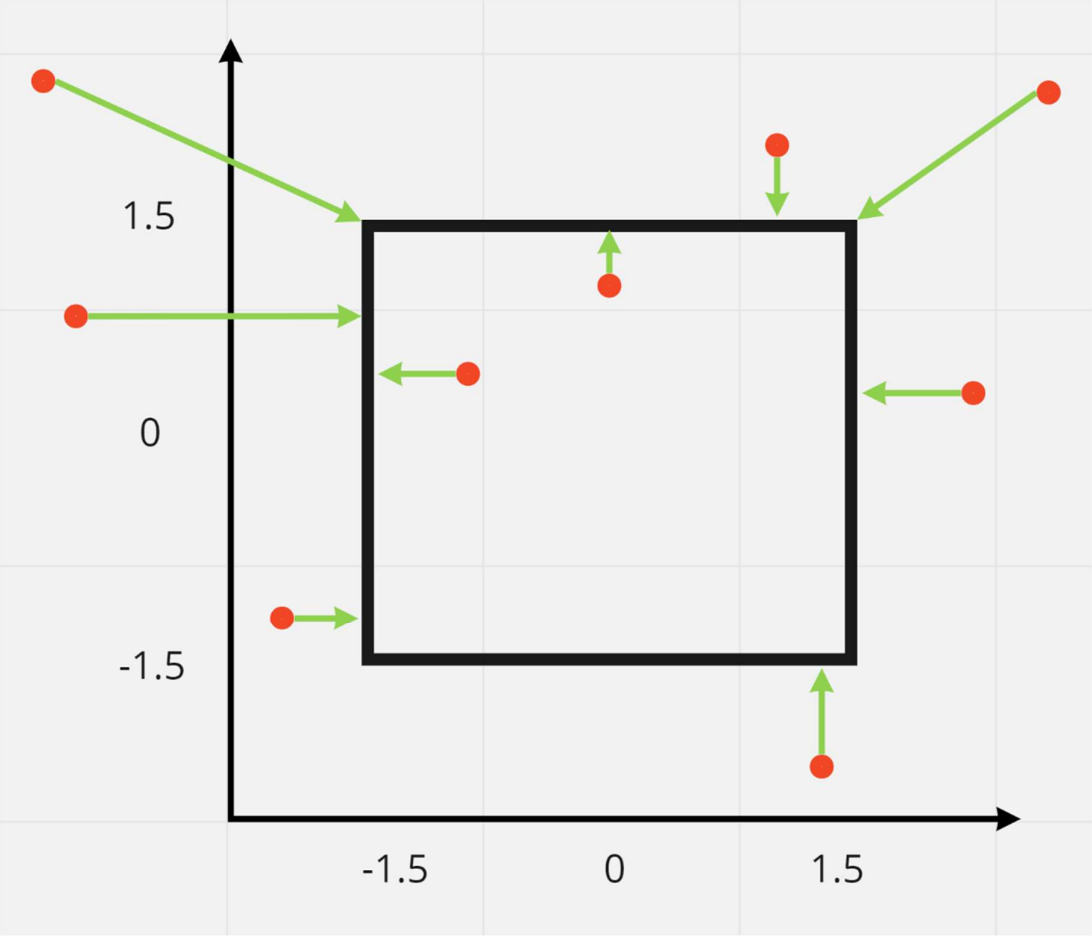
حال به سراغ دستورات لازم برای اجرا می‌رویم:

```
roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
rosrun square_control pose_monitor.py
rosrun square_control move_robot.py
```

ابتدا شبیه‌ساز و یک جهان خالی راه‌اندازی می‌شود و پس از آن گره pose_monitor راه‌اندازی می‌شود و منتظر حرکت ربات برای جمع‌آوری اطلاعات حرکت ربات می‌ماند و در نهایت هم move_robot راه‌اندازی می‌شود و ربات شروع به حرکت می‌کند و پس از ۱۰ دور پیمایش مربع مورد نظر، می‌ایستد و به گره pose_monitor اعلام می‌گردد که خطا و مسیر را نمایش دهد.

اندازه‌گیری و کمی‌سازی خطا

ایده‌ای که برای اندازه‌گیری و کمی‌سازی خطا صورت گرفته است به صورت زیر است:



این فاصله‌ای که در بالا نشان داده شده است را برای هر کدام از نقاط با توجه به شرایطی که در بالا مشخص است، محاسبه می‌کنیم و میانگین این مقادیر را به عنوان خطا گزارش می‌کنیم که در حرکت ربات ما این مقدار در دور دهم پیمایش برابر ۰.۳۷ است.

مسیر حرکت

در انتها نیز مسیر حرکت ربات به صورت گرافیکی را می‌بینیم که در ابتدا هنوز به شرایط پایدار نرسیده و خطای آن بیشتر است و در ادامه به یک حالت پایدار می‌رسد.

با توجه به اینکه سرعت ربات ثابت فرض شده و نمی‌توان هنگام چرخش فرمان کمی از سرعت خطی ربات کم کرد، باعث می‌شود که کمی از مسیر اصلی و دقیقی که مدنظر ما است فاصله بگیریم که با کمک ثابت‌هایی که قبلا توضیح داده‌شد تلاش شده این خطا خیلی زیاد نشود.

مسیر حرکت ربات در صفحه‌ی بعد مشخص شده است.