ROS Localisation and Navigation using Gazebo

Haidyn McLeod

Abstract—The ROS navigation stack is powerful for mobile robots to calculate their position and orientation so they can navigate reliably between obstacles to a goal position. Tuning the navigation stack on a real robot in the real world can be costly and dangerous. This paper presents two different robot models using the Gazebo physics simulator to tune and test the performance of the robots navigation stack.

Index Terms—Robot, Particle Filter, Kalman Filter, Udacity, Localisation, Navigation, ROS, Gazebo, Differential drive, Skid-steer.

1 Introduction

Navigating a mobile robot platform is an essential area of focus in robotics. The Robotic Operating System (ROS) provides a navigation stack that performs localisation of the robots position and orientation within a known world and path planning to produce a safe path for the robot to execute between the robots current location and its target location. The navigation stack provides various methods for achieving this with only the need to tune parameters. Each section will be discussed in detail in the background section.

1.1 Model

Effective navigation requires that an accurate robotic model is produced. The most common types of consumer-based models are based on a differential drive, where two wheels are driven independently of each other. A similar model is omnidirectional, or for tracked vehicles, a skid-steer model is required. With the increase of self-driving cars, a bicycle model is used. For farm machinery and mining equipment, an articulated model will be required.

1.2 Localisation

Localisation is a collection of state estimation nodes, each of which is an implementation of a non-linear state estimator for robots position and orientation moving in 3D space [1]. ROS provides multiple methods to perform localisation which include the various types of Kalman filters and particle filters.

1.3 Path Planning

Path planning is using the predicted state estimates with a known environment map to create a path that avoids obstacles and considers the robot dynamics to produce a smooth path. To use this node, we need to define global and local planners. ROS provides three global planner methods, *navfn*, *carrot* and *global*. The four local planner methods *base local planner*, *dynamic window (dwa)*, *elastic band (eban* and *timed elastic band (teb)*.

2 Background

The ROS navigation stack requires that the robot is configured in a particular manner. Fig. 1 shows an overview of this configuration [2]. Each requirement for the navigation stack will be discussed below.

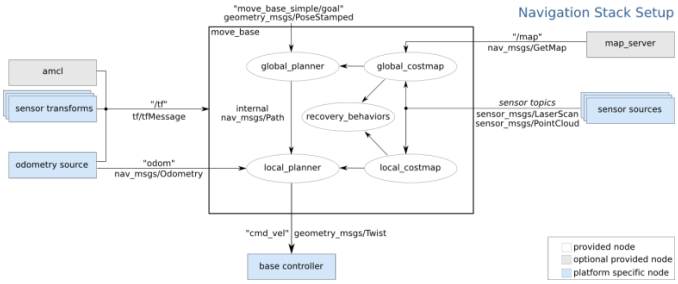

Fig. 1. ROS navigation stack layout. The white components are required components that are already implemented, the grey elements are optional components that are already implemented, and the blue components must be created for each robot platform [2].

2.1 Model

2.1.1 Differential Drive

Due to the independent nature of two drive wheels, these models are commonly used because they do not require specialised hardware and can rotate in position. They are widely used in indoor environments as they need a free rotating wheel for balance. The movement model is defined in Fig. 2.

2.1.2 Skid-Steer

Skid-steer models are often associated with tracked vehicles but are also used for multiple-wheeled vehicles that do not have any wheel group controlled by steering. An example of this is the *HUSKY unmanned ground vehicle*. The control of a skid-steer is through the left and right wheel groups. Each wheel in the group is driven together, and the two-wheel groups are operated independently of the other. This enables them to perform the same movement as the *differential drive* model. They are commonly used in rough terrain as each driven wheel provides better traction and control of the robot. The movement model is defined in Fig. 2.

The two models presented in this paper will compare a differential drive robot with a skid-steer robot.

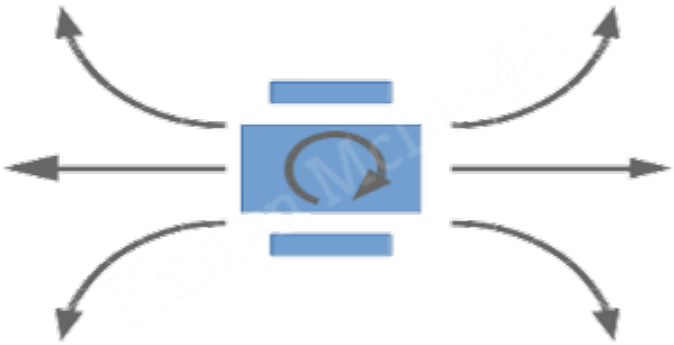

Fig. 2. Differential drive and skid-steer robot movement directions.

2.1.3 Omni Directional

These models are commonly produced in either a three-wheel or four-wheel orientation. Each wheel is independent and is placed at an angle to each other which lies tangential to a circular radius from the centre of the robot, (120°for a three-wheel model and 90°for a four-wheel model. They are commonly used in high end or research models for indoor environments as they require special wheels but can move in any direction with the need to rotate. The movement model is defined in Fig. 3.

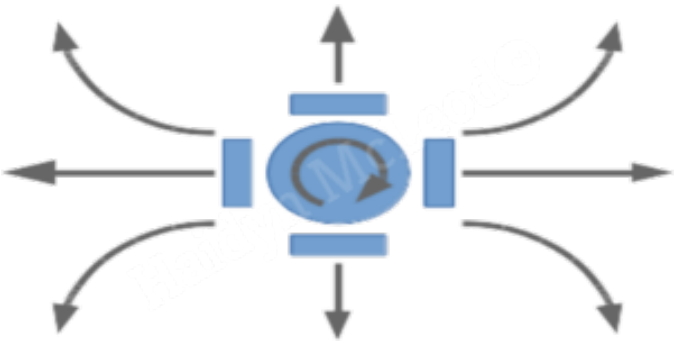

Fig. 3. Omni directional robot movement directions.

2.1.4 Bicycle model

This model is the basis of any vehicle with a steering axle group, such as a car. Either only one or both wheels are driven dependently of each other with one of the wheel groups being steered to control its direction of travel. These models require special consideration of path planning to account for their turning circles. The movement model is defined in Fig. 4.

2.1.5 Articulated model

Articulated vehicles are defined as having a pivot point between the front and rear wheels. They have the similar properties as the *Bicycle model* but the steering is controlled by changing the angle between the front and back chassis. An example of this model is the Caterpillar 730C truck. They are commonly used on heavy machinery as there are no sliding forces placed on each wheel when turning and when the pivot is centralised the front, and rear wheels will follow the same trajectory which is good for navigating tight turns. The movement model is defined in Fig. 4.

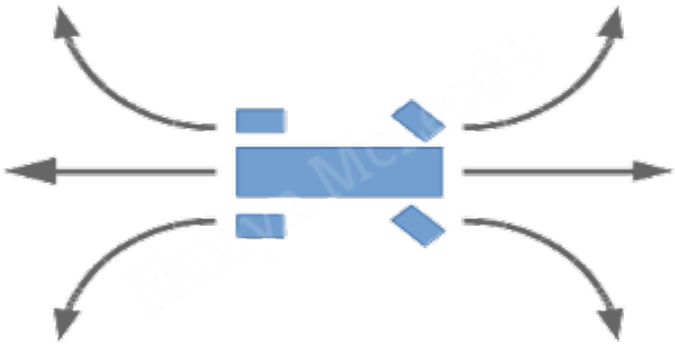

Fig. 4. Bicycle and Articulated model movement directions.

2.2 Localisation

2.2.1 Kalman Filters

A Kalman filter is an optimal estimation algorithm. It is used to estimate a systems state when it can not be measured directly, but an indirect measurement is available [3]. Kalman sensors use different measurement devices that are subject to noise or uncertainty to predict the robots state. For example the robots position, velocity and heading. There are three versions of the Kalman filter. 1. Linear Kalman filter, 2. Extended Kalman filter (EKF) and 3. Unscented Kalman filter (UKF).

The linear Kalman filter is used when the systems states are identical to the sensor measurements observed. This is why it is a linear Kalman filter. The EKF is a non-linear system state estimate. It is used when measurements are obtained from no linear sensors. It can provide accurate system state estimates when sensor measurements are only partial or different to the tracked state variables. EKF's uses a Jacobi matrix to map non-linear devices to the EKF state. In recent time another highly non-linear Kalman filter has been developed which continues as if it is a linear prediction model by using approximations to predict the robots states and covariance of highly non-linear sensors called a UKF. Both the EKF and UKF provide the same system output predictions but with two different approaches where each approach has benefits under certain conditions over the other.

2.2.2 Particle Filters

Another method of predicting the system states is to make random discrete guesses of where the robot is. As the robot moves around and takes measurements of the surrounding environment, it uses this reading to update each particles probability of the robots estimated position. Places of higher likelihood will over time collect more particles and therefore be more accurate of the robots posterior belief. In this paper, each particle will contain x, y, and heading states used to predict a robot's actual state.

Particle filters provide excellent position estimations in a known mapped environment or with complex robot models. Particle filters are also used If the robot is in one of two odd-shaped regions, or the system can behave very different in certain regions [4].

2.3 Comparison / Contrast

In a linear system with Gaussian noise, the Kalman filter is optimal. In a system that is non-linear, the Kalman filter can be used for state estimation, but the particle filter may give better results at the price of additional computational effort. In a system that has non-Gaussian noise, the Kalman filter is the optimal linear filter, but again the particle filter may perform better. The UKF provides a balance between the low computational effort of the Kalman filter and the high performance of the particle filter [5], [6].

The particle filter has some similarities with the UKF in that it transforms a set of points via known non-linear equations and combines the results to estimate the mean and covariance of the state. However, in the particle filter, the locations are chosen randomly, whereas in the UKF the points are selected with a specific algorithm. Because of this, the number of points used in a particle filter needs to be much greater than the number of points in a UKF. Another difference between the two filters is that the estimation error in a UKF does not converge to zero in any sense, but the estimation error in a particle filter does converge to zero as the number of particles (and hence the computational effort) approaches infinity [5], [6].

The work presented in this document will use a particle filter in the form of the Adaptive Monte Carlo Localisation (AMCL) method. For more on the `amcl` method in ROS, please refer to http://wiki.ros.org/amcl for more information.

2.4 Path Planning

Path planning is defined in the `move_base` node. It provides an interface for the global and local planners used in the navigation stack.

2.4.1 Global Planer

The first planner is the `navfn`. It provides a fast interpolated navigation function that can be used to create plans for a mobile base. The navigation function is computed with the Dijkstra's algorithm [7].

The `global_planner` package provides an implementation of a fast, interpolated global planner for navigation. It was built as a more flexible replacement to `navfn` [8]. The path search algorithm for the planner can be selected from the Dijkstra path, A* path, grid path or a simple potential calculation which can use a quadratic approximation.

The `carrot_planner` is a simple global planner that takes a goal point from an external user, checks if the user-specified goal is in an obstacle, and if it is, it walks back along the vector between the user-specified goal and the robot until a goal point that is not in an obstacle is found. It then passes this goal point on as a plan to a local planner or controller. In this way, the carrot planner allows the robot to get as close to a user-specified goal point as possible [9].

The Global planner presented in the project is the `global_planner`. For more on the `global_planner` method in ROS, please refer to http://wiki.ros.org/global_planner for more information.

2.4.2 Local Planer

The first of the four planners for moving the robot locally is the `base_local_planner`. This package provides a controller that drives a mobile base in the plane. This controller serves to connect the path planner to the robot. Using a map, the planner creates a kinematic trajectory for the robot to get from a start to a goal location. Along the way, the planner creates, at least locally around the robot, a value function, represented as a grid map. This value function encodes the costs of traversing through the grid cells. The controller's job is to use this value function to determine produced velocity commands to send to a mobile base [10].

The `dwa_local_planner` provides the same functionality as the DWA but is different in how the robot's control space is sampled. The `base_local_planner` samples from the set of achievable velocities over the entire forward simulation period given the acceleration limits of the robot, while DWA samples from the set of achievable velocities for just one simulation step given the acceleration limits of the robot. This means that DWA is a more efficient algorithm because it samples a smaller space [10].

The `eband_local_planner` package implements the Elastic Band method first described in 1993 at the IEEE International Conference on Robotics and Automation [11], [12]. It allows optimising a global plan locally, minimising the length of the path and keeping it away from obstacles while at the same time taking into account moving obstacles. The local planner computes an elastic band within the local cost map and attempts to follow the path generated by connecting the centre points of the band using various heuristics [12]. An advantage of this approach is that they optimise the plan incrementally. That is, the longer the robot moves, the better the trajectory path will be. It is designed for use with omnidirectional and differential drive robots.

The `teb_local_planner` or Timed Elastic Band locally optimises the generated global plan concerning minimising the trajectory execution time, separation from obstacles and compliance with kinodynamic constraints such as satisfying maximum velocities and accelerations at runtime [13]. The `teb_local_planner` takes into consideration detailed robot configurations such as turning circle and wheelbase. It is designed for robots that have a turning circle such as a car or articulated robot.

The discussion section will detail test of each planner. The local planner used in the model is the `eband_local_planner`.

3 SIMULATIONS

3.1 Differential-Drive Model

3.1.1 Model design

The Differential-drive model consists of two independent drive wheels with caster wheels at each end for balance. Perception for the navigation stack is through a 180°2D laser scanner positioned 150mm forward of the centre of rotation. An onboard camera is added for visual purposes. The model's parameters are 400 x 300 x 250mm (LxWxH), weight is 25kg, and wheel radius is 200mm. Fig 5 shows the final model design.

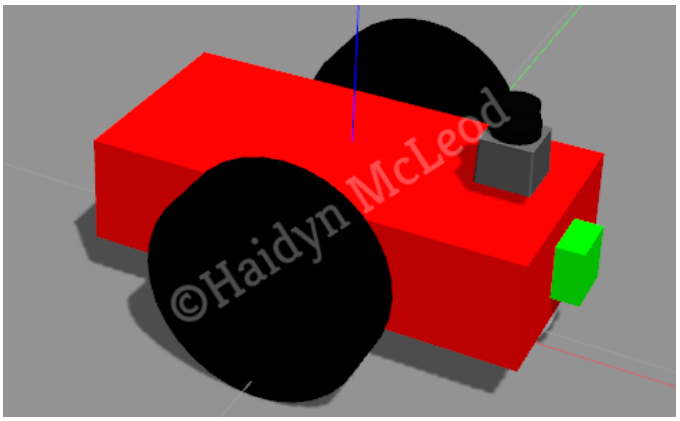

Fig. 5. Differential drive model used for testing.

3.1.2 Packages Used

Both the differential-drive and skid-steer models use the `amcl` package for localisation, `global_planner` for the global planning and `eband_local_planner` as the local planner. These three provided the best results through testing an will be discussed in the results section. The drive controller uses the `differential_drive_controller` with an update rate of 10, torque of 10 N.m, receives `/odom` topics and publishes motor commands to the `/cmd` topic.

3.1.3 Parameters

The ROS particle filter package used for localising the robot is `amcl`. `amcl` has three parameter groups to consider. Localisation, Laser, Odometry. Parameters were tuned both through the use of the ROS amcl wiki [14] and `rosrun rqt_reconfigure rqt_reconfigure`.

The localisation parameters were largely defined as their default values. The translational `update_min_d` and rotational `update_min_a` update distances were decreased to 0.1m and 0.13 radians respectively for more frequent positional predictions. The number of minimum particle `min_particles` was lowered to 50 and the max `max_particles` was lowered from 5000 to 500. This was largely due to limiting the computational burden on the system. The number of filter updates required before resampling `resample_interval` was decreased to 1 to increase the accuracy of the particle predictions.

The laser parameters were unchanged from their default values.

Odometry model type `odom_model_type` was changed to `diff-corrected` to correct the Gaussian re-sampling bug [15]. The expected noise in odometry rotation estimate from the rotational component of the robot's motion `odom_alpha(1-4)` is decreased to 0.05 across all as the model has negligible wheel slip due to the model's centre of inertia having no offset from the drive wheels and the casters providing omnidirectional slip. The remaining parameters are the default.

The controller and planner frequency of the path planners `move_base` node is decreased to 10Hz. This is to reduce computational load as the robot is slow moving. The global path planner used is `global_planner`. It is defined with default values. The local planner uses the elastic band method `eband_local_planner/EBandPlannerROS` with the parameters defined in Table 1.

TABLE 1
EBandPlannerROS parameters

Parameter	Value	Description
xy_goal_tolerance	0.1	Distance tolerance
yaw_goal_tolerance	0.15	Orientation tolerance
rot_stopped_vel	0.05	Angular velocity min
trans_stopped_vel	0.05	Linear velocity min
max_vel_lin	1.5	Max linear velocity
max_vel_th	1.0	Max angular velocity
min_vel_th	0.05	Min angular velocity
min_in_place_vel_th	0.1	Min in-place angular
max_translational_acceleration	0.05	Linear acceleration
max_rotational_acceleration	0.1	Angular acceleration
differential_drive	true	Diff-drive mode
max_acceleration	0.1	Maximum acceleration
k_prop	4.0	P gain of PID
k_damp	3.5	D gain of PID
k_int	0.001	I gain of PID

Note: The elastic band method requires installation `sudo apt-get install ros-kinetic-eband-local-planner`.

3.2 Skid-steer Model

3.2.1 Model design

The second model is a skid-steer platform. This platform is used as it provides the same movement model, Fig. 2 but requires different control and model dynamics. This will enable skid-steer of system parameters between model types. Model sensors are identical to the differential-drive model, but the laser scanner is located at the centre of rotation of the robot. The model's parameters are 460 x 460 x 300mm (LxWxH), weight is 14kg, and wheel radius is 160mm. Fig 6 shows the final models design.

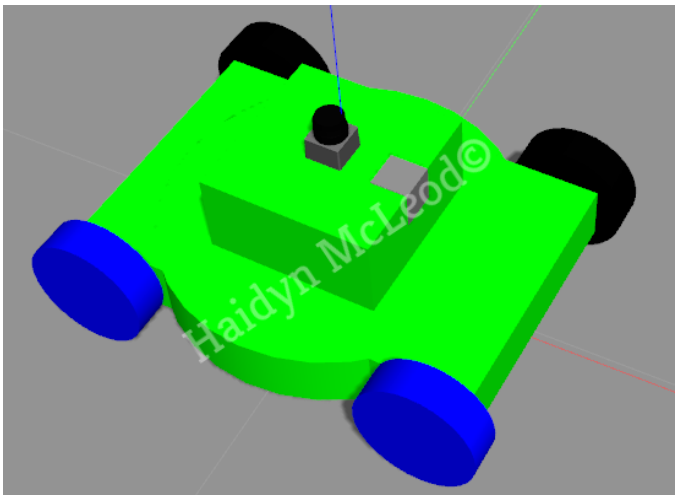

Fig. 6. Skid-steering model used for testing. Blue wheels indicate right pair drive wheels and black left pair drive wheels.

Each wheel was modelled to have friction and slip properties in both the x and y-axis to allow for the rotational slip. Table 2 describes the slip properties.

TABLE 2
Skid-steer wheel friction and slip properties.

mu	1.0	Rotation friction factor
mu2	1.0	Translation friction factor
slip1	0.5	Rotation slip factor
slip2	0.0	Translation slip factor

3.2.2 Packages Used

The navigation stack parameters are the same as the differential-drive model, but the drive controller uses the `skid_steer_drive_controller` with an update rate of 100, torque of 200 N.m, receives `/odom` topics and publishes motor commands to the `/cmd` topic. The controller covariance for x and y is also incensed to 0.001.

3.2.3 Parameters

The localisation parameters and laser parameters were unchanged from differential-drive model defined above with `odom_model_type` changed to `diff`. The odometry noise in the rotational component of the robot's motion `odom_alpha(1-4)` was also increased due to skid-steer slippage and a friction bug in Gazebo 7. The bug uses a friction pyramid which creates singularity when the bots rotation aligns with the x or y-axis. The extra rotation noise is required as Gazebo cannot rotate the robot about its centre of inertia. In newer versions of Gazebo, the problem has been fixed, but the paper will simulate in the default kinetic Gazebo version.

TABLE 3
odom_alpha(1-4) expected noise in odometry estimate from the rotational component of the robot's motion.

odom_alpha1	0.5	Rotation noise from rotational move
odom_alpha2	0.4	Rotation noise from translational move
odom_alpha3	0.1	Translation noise from translational move
odom_alpha4	0.4	Translation noise from rotational move

The controller frequency of the path planners `move_base` node is increased to 50Hz. This is to help reduce the impact of the rotation bug mentioned above. The global path planner used is `global_planner`. It is defined with default values. The local planner uses the elastic band method, as defined in the differential-drive model, with the updated parameters defined in Table 4.

4 Results

4.1 Localisation Results

The localisation of the two robots starting position and goal location were identical. The test environment is shown in Fig 7. The initial poses mean value prediction of the robots is zero with covariance of 1.0.

4.1.1 Differential-Drive Model

The differential-drive robot initial localisation is randomly noisy with respect to the map origin. Fig. 8 shows the initial prediction location relative to the robot.

TABLE 4
EBandPlannerROS parameters

Parameter	Value	Description
xy_goal_tolerance	0.15	Distance tolerance
yaw_goal_tolerance	0.2	Orientation tolerance
max_vel_lin	0.75	Max linear velocity
max_vel_th	1.0	Max angular velocity
min_vel_th	0.05	Min angular velocity
min_in_place_vel_th	2.0	Min in-place angular
max_translational_acceleration	2.5	Linear acceleration
max_rotational_acceleration	1.5	Angular acceleration
max_acceleration	2.5	Maximum acceleration
k_prop	1.0	P gain of PID

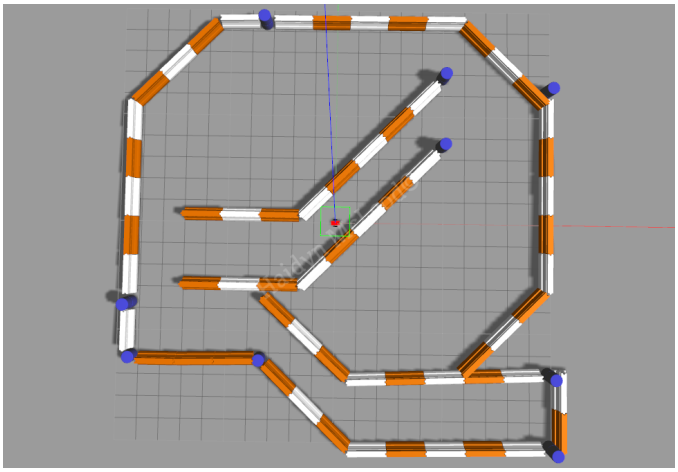

Fig. 7. Test environment.

The localisation of the robot to the goal location begins to converge on the position of the robot within a movement 0.5m from the starting position. Fig. 9 shows the particle conversion at 0.5m distance travelled.

At the 3m point, the particles have the correct robot orientation but a high covariance in the positional prediction. Fig. 10 shows the particle conversion at 3m distance travelled.

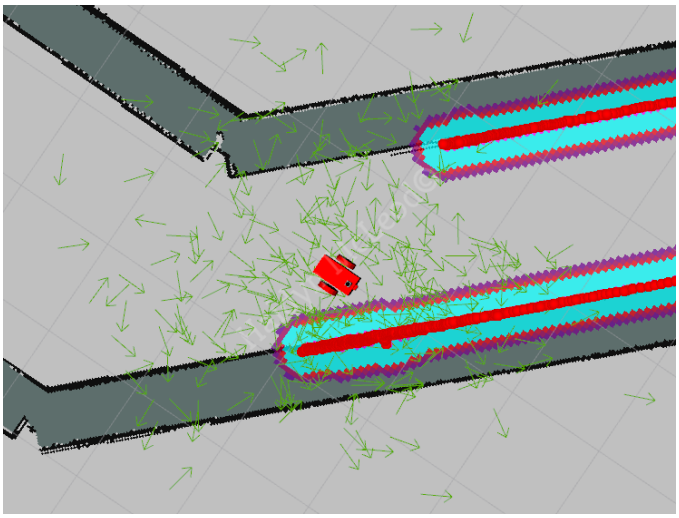

Fig. 8. Robot's starting location and particles prediction.

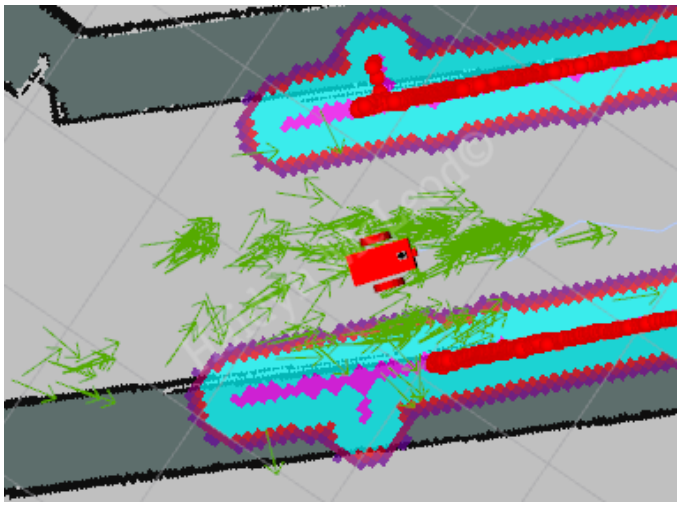

Fig. 9. The particles prediction of the robot at 0.5m from the start location.

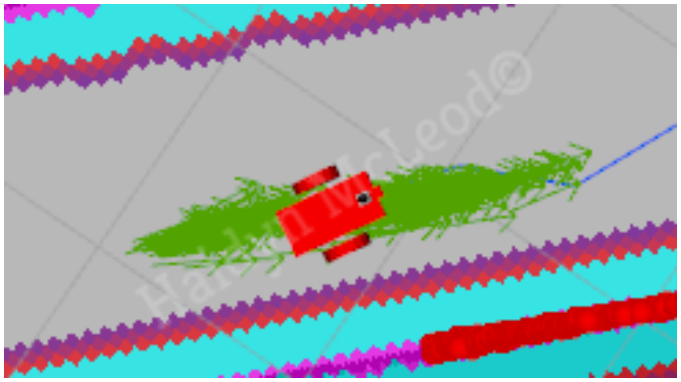

Fig. 10. The particles prediction of the robot at 3m from the start location.

At the end of the first corridor or 5.5m travelled, the laser has detected the boundary wall of the test environment, enabling the `amcl` node to localise the robot accurately. Fig. 11 shows the particle conversion at 5.5m distance travelled.

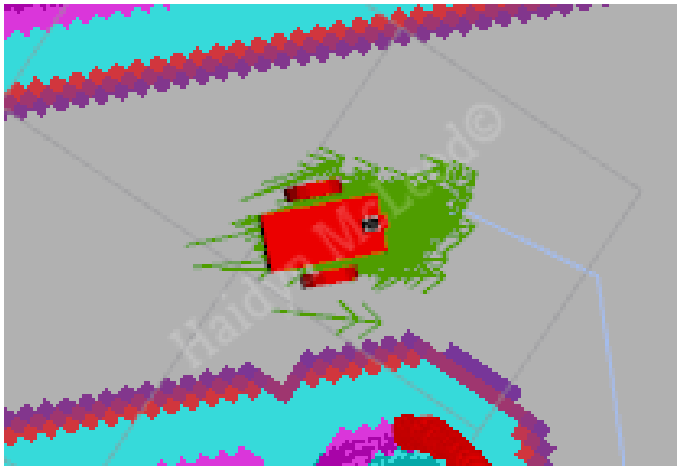

Fig. 11. The particles prediction of the robot at 5.5m from the start location.

Between the 5.5m travel distance and the goal location, no significant data or predictions were observed. The final goal prediction of the robot is shown in Fig. 12 shows the predictions at the goal location.

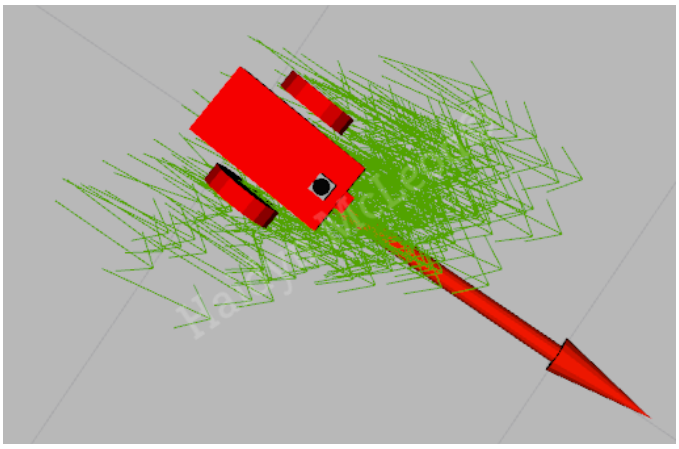

Fig. 12. The particles prediction of the robot at the end location.

The following of the Global path was observed to follow consistently. The global path, however, was not a smooth prediction when avoiding obstacles. The path produced sharp turns and fast deceleration of the robot when travelling at speed. This sudden deceleration, when moving at high speeds, caused the robots inertia to pitch the model forward and the laser scanner would detect the ground as an obstacle. This had two effects. 1. The robot would stop, and a new global path around the false obstacle is calculated and 2. the particle predictions decreased accuracy momentarily. This error is due to the model design and the three points of contact with the environment surface. This is discussed in the discussion section.

4.1.2 Skid-Steer Model

Fig. 13 shows the initial prediction location relative to the robot.

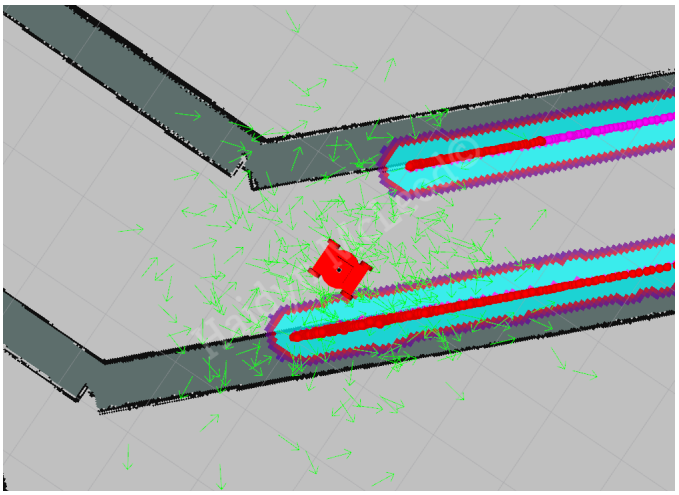

Fig. 13. Robot's starting location and particles prediction.

As with the differential-drive robot, a movement 0.5m from the starting position begins to converge on the position of the robot. Fig. 14 shows the particle conversion at 0.5m distance travelled.

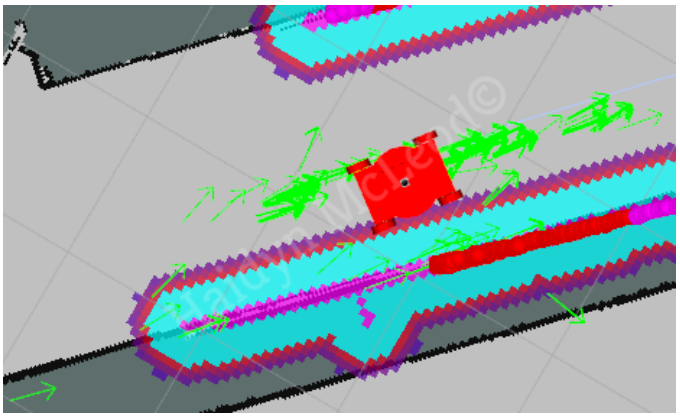

Fig. 14. The particles prediction of the robot at 0.5m from the start location.

At the 2m point, the particles have an accurate prediction of the robots position and orientation. Fig. 15 shows the particle conversion at 2m distance travelled.

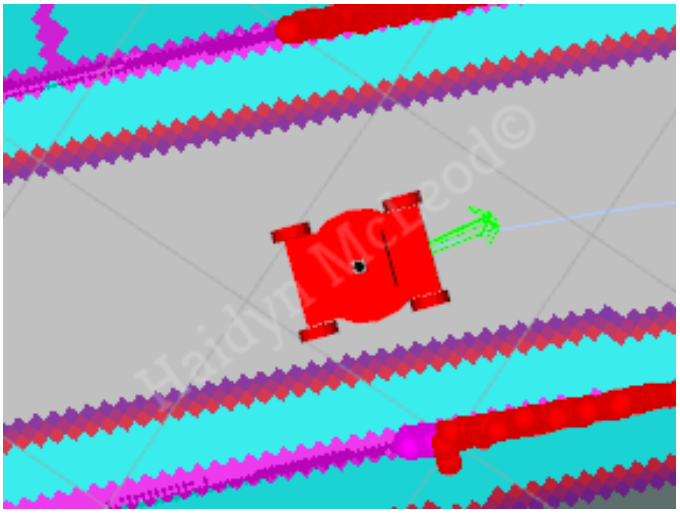

Fig. 15. The particles prediction of the robot at 2m from the start location.

At the end of the first corridor or 6m travelled, the particles have accurately predicted the location of the robot. Fig. 16 shows the particle with the robot model removed for clarity.

The final goal prediction of the robot is shown in Fig. 17 shows the predictions at the goal location.

The following of the Global path was observed to be inconsistent when navigating turns. This is due to the Gazebo simulator singularity error mentioned above. This is evident in Fig. 16. This can also cause the robot to continually rotate and move around the goal location, as the robot can only exit the singularity when the model transnationally drifts off the aligned axis.

4.2 Technical Comparison

From the results observed above the skid-steer model provided increased prediction accuracy over the differential drive model. Though testing the odometry model type `odom_model_type` was the cause of this inconsistency. The

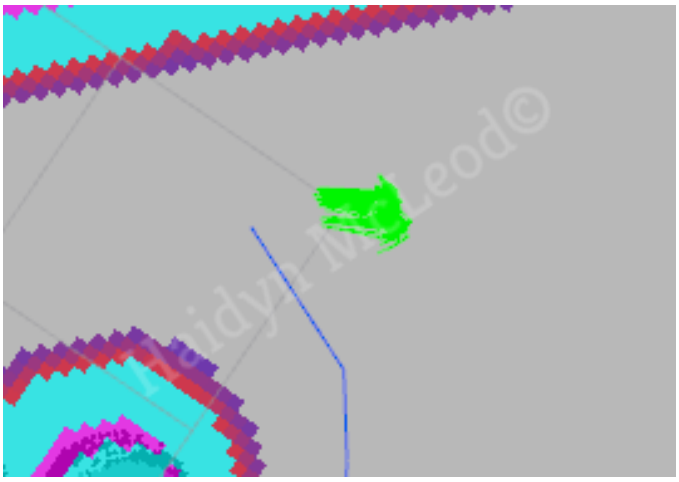

Fig. 16. The particles prediction of the robot at 6m from the start location. Robot model removed for clarity.

Fig. 17. The particles prediction of the robot at the end location.

differential drive used `diff-corrected` and the skid-steer `diff`.

The differential drive model offered more control than the skid steer and was more reliable in performing path following due to the centre of inertial having no positional offset from the drive wheel location. This will be discussed in more detail in the next section.

The skid-steer model does provide better movement stability due to the four evenly positioned points of contact with the ground. The differential drive model does have a caster wheel on each end, but the rear caster is smaller than the front, to provide better driveability, creating a three-point contact and potential rocking motion.

5 Discussion

During the testing of various models and navigational stack observations were observed. First, Performance and controllability of the model are heavily dependant on the model's design. A model was tested that used the differential drive set-up, but the drive wheels were placed at the back of the model and caster wheels at the front, thus offset from the inertia's position. The result was the robot struggled

to perform any rotation, even when the drive torque was substantial. When rotation was observed sliding, and a skidding motion eventuated from the drive wheels resulting in incorrect and inconsistent odometry readings. This was also the case for the skid-steer model when the width of the robot was less than the wheel-length.

The model design was observed to have a minimal effect on `amcl` localisation. The most significant impact is the laser sensors position on the robot. However, this is also negligible when compared with the odometry model type of `diff-corrected` which provided the highest prediction errors. When testing both models with the same odometry model type of `diff`, it has been observed that the differential drive model offered better particle prediction. This is most likely the case due to the more unpredictable odometry slip in the skid-steer model.

Overall, `amcl` works well in a short distance of travel. It does, however, have problems when the world is not mapped or if there is no obstacle seen by the laser scanner. Over time, a robot in open spaces will drift both transnationally and rotationally. This will cause the particles uncertainty to increase and create an unknown robot position. Another potential problem is with the kidnapped robot problem. `amcl` will cause the robot to behave unpredictable, potentially crashing until it has relocated itself in the known world. However, `amcl` has built in recovery behaviours that reset the cost maps, thus preventing old commands and maps from influencing the behaviour of the robot.

6 Conclusion / Future work

The paper compares the ROS navigation stack with two different mobile based robot platforms in a simulation. Various ROS `amcl` node parameters are used to predict the location of a differential-drive and skid-steer models accurately. `amcl` provides fast and accurate predictions that are minimally influenced by the platform's design. Although `amcl` and `move_base` provides a large number of parameters to tune, It has been shown that with very few parameter adjustments, robot localisation can be performed on different robot platform types.

Future extensions of this work could include dynamic obstacles in the test environment. Larger and more complex environments. Other models such as omnidirectional and steering based models. A 360 laser scanner could also provide better and quicker localisation. The approaches taken are intended to only show the capabilities and limitations of the navigation stack between different robot platforms.

References

[1] T. Moore and D. Stouch, "A generalized extended kalman filter implementation for the robot operating system," in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*, Springer, July 2014.

[2] G. E. Eitan Marder-Eppstein, "Setup and configuration of the navigation stack on a robot," in *http://wiki.ros.org/navigation/Tutorials/RobotSetup*, ROS Wiki, August 2015.

[3] MATLAB, "Understanding kalman filters, part 1: Why use kalman filters?," in *https://www.youtube.com/watch?v=mwn8xhgNpFY*, Youtube, January 2017.

[4] J. Reich, "What is the difference between a particle filter and a kalman filter?," in *https://www.quora.com/What-is-the-difference-between-a-particle-filter-and-a-Kalman-filter*, Quora, January 2015.

[5] D. Simon, *Optimal state estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley & Sons, 2006.

[6] G. Dontas, "What is the difference between a particle filter (sequential monte carlo) and a kalman filter?," in *https://stats.stackexchange.com/questions/2149/what-is-the-difference-between-a-particle-filter-sequential-monte-carlo-and-a*, StackExchange, August 2010.

[7] c. j. Kurt Konolige, Eitan Marder-Eppstein, "navfn," in *http://wiki.ros.org/navfn*, ROS Wiki, October 2014.

[8] N. L. David Lu, "global_planner," in *http://wiki.ros.org/global_planner*, ROS Wiki, January 2018.

[9] c. N. L. Eitan Marder-Eppstein, Sachin Chitta, "carrot_planner," in *http://wiki.ros.org/carrot_planner*, ROS Wiki, January 2018.

[10] c. R. S. Eitan Marder-Eppstein, Eric Perko, "base_local_planner," in *http://wiki.ros.org/base_local_planner*, ROS Wiki, March 2018.

[11] S. Quinlan and O. Khatib, *Elastic Bands: Connecting Path Planning and Robot Control. vol 2. pp. 802-807*. Proc. IEEE International Conference on Robotics and Automation, Atlanta, Georgia, 1993.

[12] P. K. F. W. Christian Connette, Bhaskara Marthi, "eband_local_planner," in *http://wiki.ros.org/eband_local_planner*, ROS Wiki, January 2018.

[13] J. H. Christoph Rsmann, "teb_local_planner," in *http://wiki.ros.org/teb_local_planner*, ROS Wiki, December 2016.

[14] V. S. Brian P. Gerkey, contradict@gmail.com, "amcl," in *http://wiki.ros.org/amcl*, ROS Wiki, July 2017.

[15] h. chadrockey, "Amcl mixes units when sampling the gaussian distribution #20," in *https://github.com/rosplanning/navigation/issues/20*, ROS Wiki, February 2013.