Navigation with the LaserSharkCatBot

James Morris (AwokeKnowing)

Abstract—One of the barriers to home robotics is perception. The perception is that the more advanced a robot is, the more boring, slow, expensive, and detached from reality it will be. To combat this perception we introduce LaserSharkCatBot an advanced meme-dense robot concept, with full autonomous navigation, for home and commercial indoor environments. In this paper we focus on the navigation capability, showing the components necessary for successful indoor point-to-point navigation.

Index Terms—Robot, IEEEtran, Udacity, LaTeX, Localization.

1 Introduction

With the popularity of robots, sharks with lasers, cats chasing lasers, as well as the more niche trend of 'furries', which involves appearing in public with animal costumes, we blend it all together with the goal of developing an emotional support robot for the digital-native generation. This robotic platform, the LaserSharkCatBot is envisioned to be versatile enough to appeal to a wide variety of people, and it's novel appearance could help deflect expectations as to what it's behaviors should comprise.

Once such a platform goes viral and achieves mass adoption, software developers would be able to add 'apps' and otherwise commercialize the behaviors. With it's pet-like profile, the bot could follow consumers and learn their preferences, and with the shark-like forward appearance, the bot could potentially achieve an abnormally high level of personal interactions, for example, by playing suspenseful music and charging toward the consumer.

It is crucial therefore, that the platform have excellent home navigation capabilities. Here we investigate the effectiveness of a differential-drive based robot navigating using Adaptive Monte-Carlo Localization (AMCL).

2 Background

A number of different methods of localization have been developed. Broadly, they can be grouped into those that are given a global map a priori and must locate the robot on the map, and the SLAM-based algorithms which dynamically build a map. Here we investigate the AMCL (non-SLAM) method. This method has the benefit of being quite stable once a reliable map has been generated by other means.

2.1 Kalman Filters

The Kalman filter algorithm applies a more analytical approach to localize a robot given the odometry of the robot. It performs a normalized weighted sum of the predicted location based on past observations, and the newly observed location based on odometry. Kalman Filters are often used to fuse sensor data into an updated internal representation of the location and pose of a robot within a map. The filter basically receives new information about translation or rotation movement which the robot has just completed, and then predicts what the sensors will read. The difference in what the sensors read and what the internal model expected them to read is calculated. Then a weighting factor is used to update the internal state toward the measured state. In Extended Kalman Filters, the difference is calculated with a nonlinearity, allowing the updates to better fit the curve of the accuracy of the sensors.

2.2 Particle Filters

While the Kalman Filter algorithm can be efficient for low-compute platforms, it tends to not be robust in more dynamic environments, failing to converge and maintain the location. Therefore we choose the AMCL algorithm to manage the task of localizing the robot. The AMCL algorithm essentially starts with many 'virtual copies' of the robot in random locations on the map.

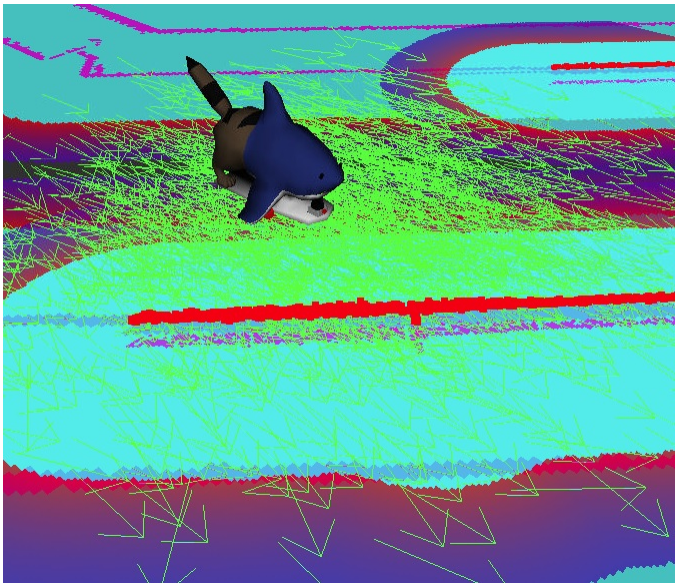

Fig. 1. AMCL Initial Particles

When the robot moves, each 'possible robot location' checks the sensor readings and compares that against what it would have predicted given the motion update. For each 'possible position' (called particle) of the robot, it can be determined how close the sensor reading was to what

would be expected if that was the actual position on the map. Particles with unlikely locations are discarded and replaced with new particles closer to the probable location.

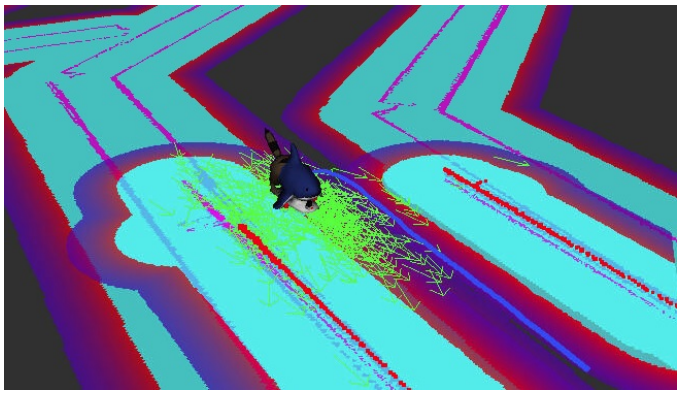

Fig. 2. AMCL Particles Converging

As the density of particles increases, the total number of particles lowers, to save computation. A minimum number of particles is kept to represent the uncertainty and to make the navigation more robust to noise, slippage, and unexpected readings.

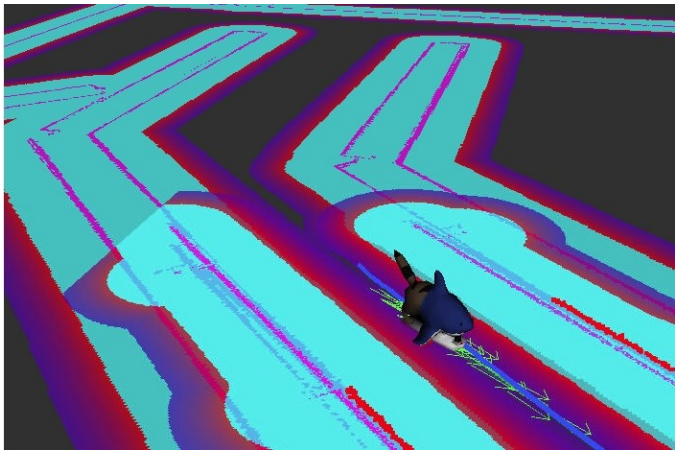

Fig. 3. AMCL Dynamic Particle Count Reducing

3 Results

It was found empirically that AMCL, with properly tuned parameters, consistently converged to to the correct localization after a few seconds of movement. Initially, a rudimentary robot was constructed consisting of a rectangular base with 10 cm wheels in a differential drive configuration.

Casters where placed on either side of the powered wheels for stability on smooth surfaces. A simulated hokuyo laser scanner, and simulated web cam were placed on top. Following parameter tuning, and using the ROS navigation stack, and AMCL implementation, the robot consistently navigated to the goal location.

Following this successful result on the mock-up robot, we tested the LaserSharkCatBot simulated robot and achieved similar results with the same parameters, even though the shape, size, and configuration of sensors were significantly different.

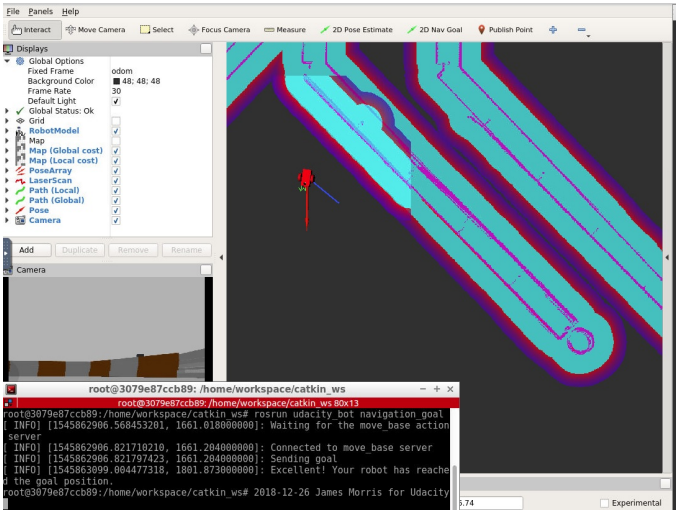

Fig. 4. Navigation Goal Reached with Test Robot

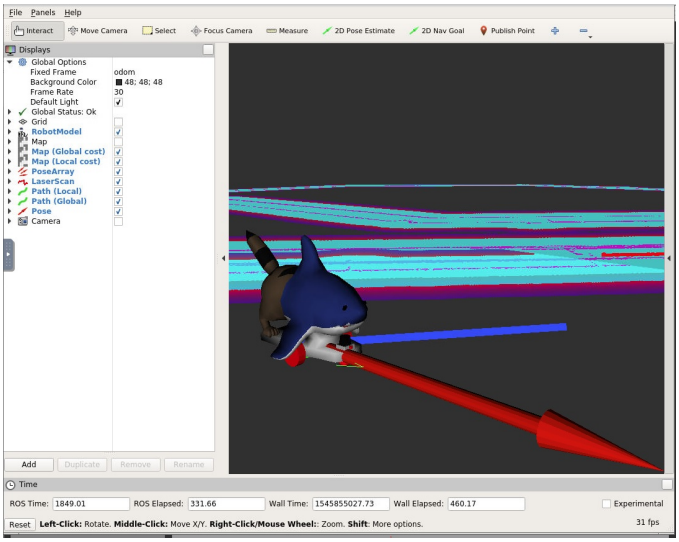

Fig. 5. Navigation Goal Reached with LaserSharkCatBot

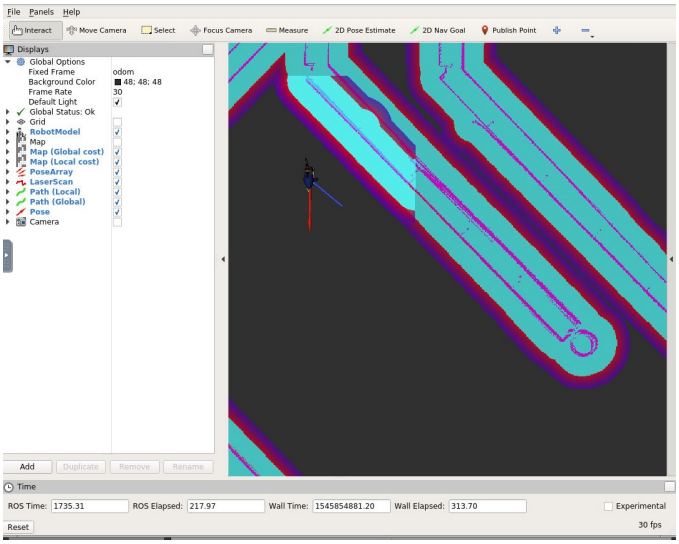

Fig. 6. Navigation Goal Reached with LaserSharkCatBot Top View

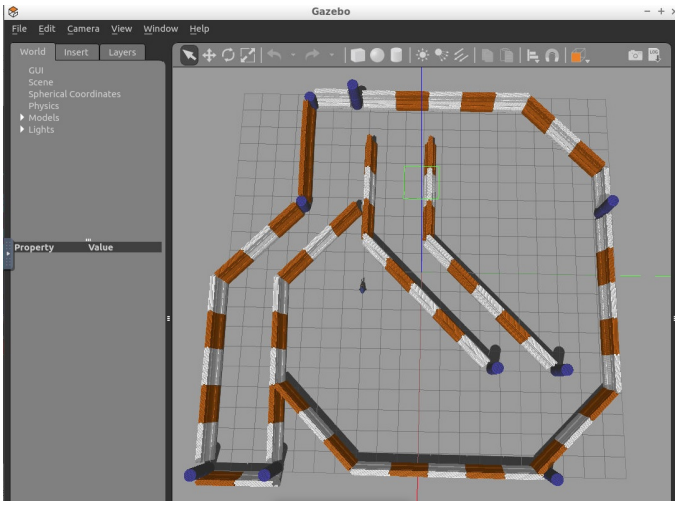

Fig. 7. Navigation Goal Reached with LaserSharkCatBot (Gazebo)

With the parameters and navigation algorithm robust across a range of platform configurations, there is a reasonable expectation that the algorithm would work on a physical robot, given sensors with similar reliability in measurements.

3.1 LaserSharkCatBot

The platform was modified to have significantly smaller wheels, and a much longer base, supporting the LaserCatSharkBot character shell. The sensors, hokuyo and camera, were placed at the front of the base.

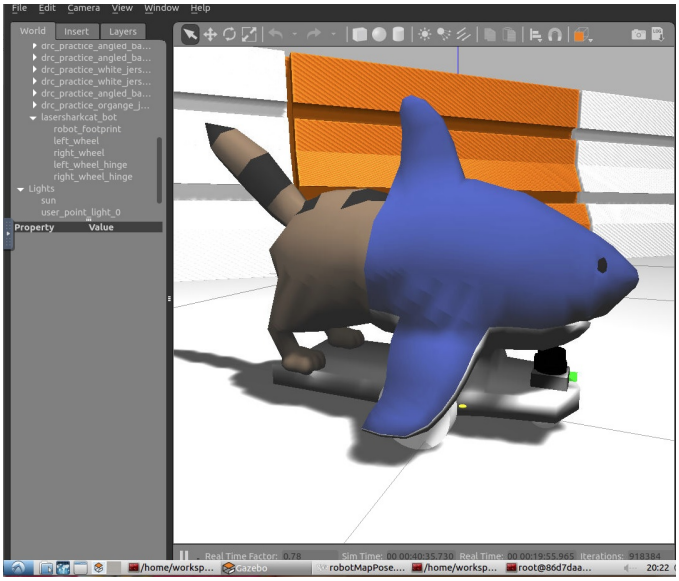

Fig. 8. LaserSharkCatBot Model (Gazebo)

The mesh was built in Blender (https://blender.org), based on the open source cat by ”Marchen” on Blend Swap (https://www.blendswap.com/blends/view/91990)

The back of this cat was removed (with permission) and attached to a partial shark model based on the one shared by ”ssatya” on Blend Swap (https://www.blendswap.com/blends/view/78052).

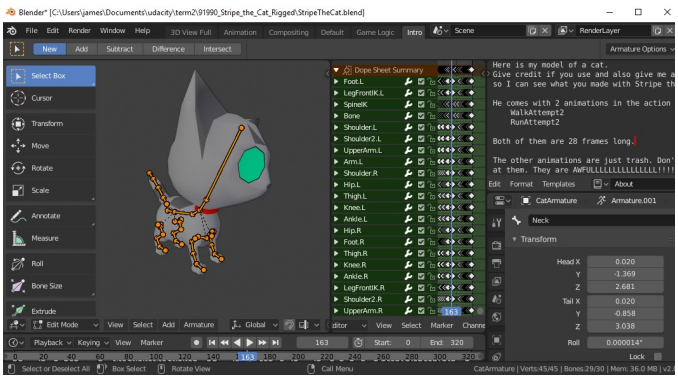

Fig. 9. Cat Model Reference

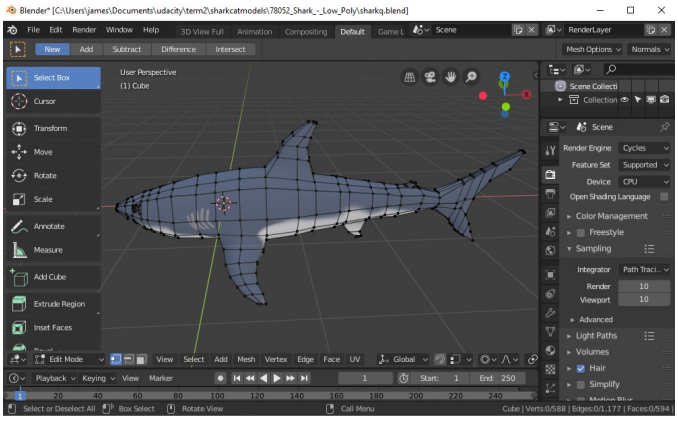

Fig. 10. Shark Model Reference

The final model’s mesh was reworked for the application, making the shark match the proportions of the cat, suggesting the cat partaking in a furry fantasy as shark with (hokuyo)lasers, simulating an endless source of entertainment for the cat. The textures were converted to vertex colors, and a base modeled for the LaserSharkCatBot. The final model was exported to a .dae (Collada) file.

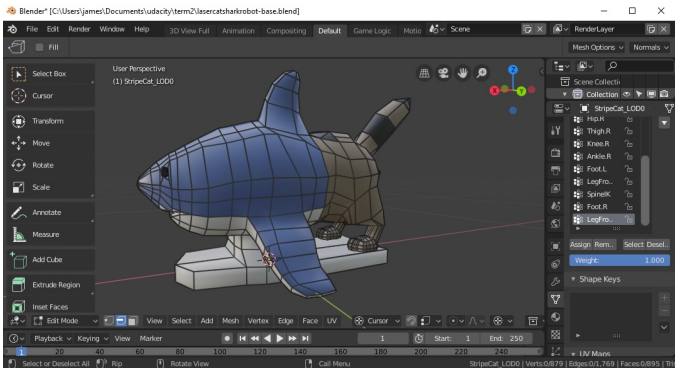

Fig. 11. Final Blender Model

The wheels on the LaserSharkCatBot were reduced to half the size of those on the test bot, and were placed closer together, to fit under the shark fins. The casters were reduced in size and repositioned further apart, for pitch stability. The base was elongated and flatted, with a supporting protrusion serving to support the character shell.

3.2 The Navigation Model

The following parameters were used to initialize AMCL.

min_particles	40	retain some uncertainty for noise
max_particles	1500	converge quickly with high likelihood
update_min_d	.09	travel at least 9cm before updating
update_min_a	.3927	rotate at least pi/8 before updating

The AMCL algorithm takes as input a global map grid. The 2d base map is generated in simulation automatically from the world. Based on that map, a cost map is generated according to specified parameters. The following are the parameters shared by the local and global cost maps.

obstacle_range	2.5	ignore further than 2.5 meters
raytrace_range	3.0	further than obstacles, to clear
transform_tolerance	.2	only recalculate if moved 20 cm
inflation_radius	.76	inflate cost near obstacle
cost_scaling_factor	3.5	smooth scale cost from zero to max

The following notable parameters are specific to the global cost map.

update_frequency	5	update the map at 5 times per second
publish_frequency	5	publish the map at 5 times per second

The following parameters are specific to the local cost map.A rolling window cost map and width/height/resolution determine the window size used to do actual local navigation.

update_frequency	5	update only 5 times per second
publish_frequency	5	publish only 5 times per second
width	5	smaller size for the local cost map
height	5	smaller size for the local cost map
resolution	.01	finer resolution the for local cost map

The following notable parameters are related to the trajectory planner.

max_vel_x	.15	ensure not saturating wheel speed
sim_time	6	ensure time for smooth path
controller_frequency	9	low enough to run with a steady tick

4 Discussion

Successful results were achieved by a careful tuning of the parameters. The key parameters were found to be max_vel_x, sim_time, inflation_radius, and cost_scaling_factor.

Specifically, initially the robot would make a good local plan but not follow it. If the plan called for a curve, the robot would just go straight. The defect was found to be the max_vel_x parameter. With its default values, the planner would plan a curve with velocities so high that both of the wheels ended up saturated at full speed and no difference in velocity was applied, forcing the robot to continue in a straight line until it collided.

The sim time turned out to be key for the robot to make smooth curves around the barriers. With a smaller sim_time, the planner tended to make very sharp curves to 'reach' the goal within the simulated time. With longer sim_time, the robot was able to take a more steady curve which properly avoided the high cost areas yet still reached the goal.

The inflation_radius needed to be adjusted as wide as possible without 'closing off' a narrow corridor. When the inflation_radius was too high, the robot would take a very long round about path across 'zero cost' territory when a short path through the corridor was possible. With the inflation radius set to even just a small gap of zero cost path, the global planner would correctly choose to go through the corridor.

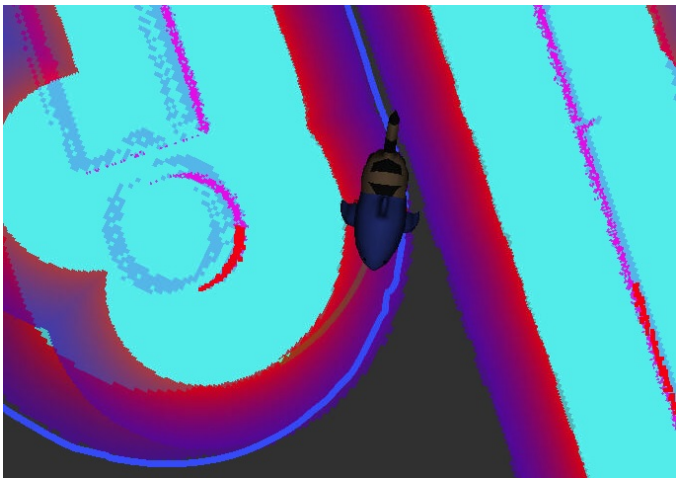

Fig. 12. Inflation Leaves Narrow Path

Similarly, lowering the default cost_scaling_factor helped the local planner to have a smooth ramp up of the cost near barriers. This was critical for forming stable paths which more smoothly moved away from barriers, and also more smoothly followed the radius of a curve without getting caught at the barriers. A higher number wouldn't give enough time between updates of crossing into cost territory and reaching the 'lethal zone'.

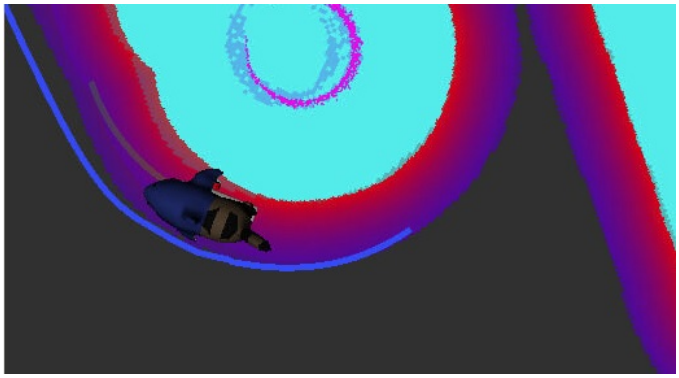

Fig. 13. Following Low Cost Path

Free-roaming robots will typically encounter dynamic environments, as well as be picked up and moved around. While the current navigation setup performed well in simulation for the point to point scenario, it would be helpful to integrate a hybrid SLAM algorithm which takes a global map as an option, but is also able to create new maps based on the environment. In the 'kidnaped robot' scenario AMCL alone would have trouble getting moved from one location to a similar one. GPS sensors could assist with this issue.

5 CONCLUSION / FUTURE WORK

These results described work done on the navigation stack of the LaserSharkCatBot, a robotic platform concept that incorporates well-known internet-rooted playfullness, with enough familiarity for consumers to be comfortable with it, and enough novelty to break free of pre-conceptions about its capabilities. If such an AI powered robot were to become produced and accepted, it could provide for many positive interactions, and advance low-cost commercial robotics.

The current navigation could be improved by further refinement of settings to allow the local planner to plan reverse motions in tight situations, rather than rely on the recovery behaviors. Another avenue for improvement would be additional bump sensors on the fins and back of the robot to avoid collisions in a more robust way. Practically, creating the physical robot and testing the navigation would likely provide the best information on how to further develop the navigation stack for the LaserSharkCatbot.

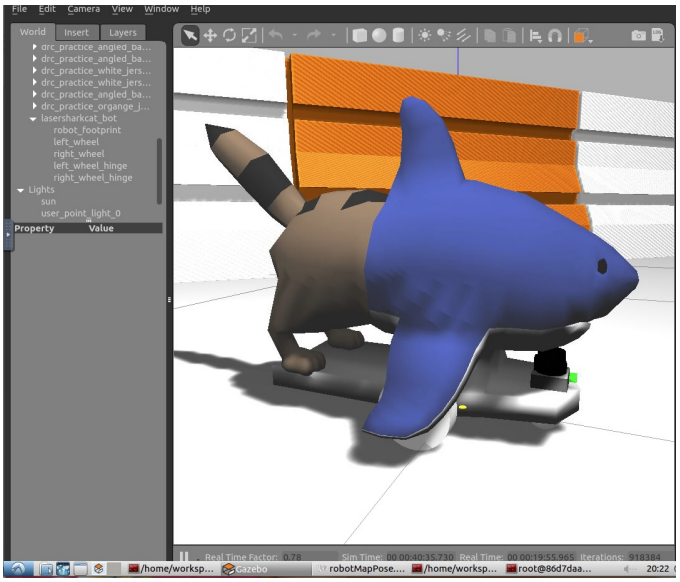

Fig. 14. LaserSharkCatBot FTW!