International Journal of Advanced Trends in Computer Applications
www.ijatca.com

Simulation of an autonomous vehicle localization

[1]Sakthi Karthik B, [2]Sundar Ganesh C S
[1]Department of Robotics and Automation, PSG College of Technology
Coimbatore, Tamilnadu
Student
[2]Department of Robotics and Automation, PSG College of Technology
Coimbatore, Tamilnadu
Assistant Professor
[2]sundarganeshsankaran@gmail.com

Abstract: *The main goal of this project is to simulate the localization of a car-like autonomous vehicle when the Global Positioning System on the vehicle fails. GPS is an important component that helps in the locating the position of the vehicle in the world. Sometimes the GPS may fail to lead to accidents or erratic motion in the case of autonomous vehicles. An effective methodology is required to locate the vehicle in the world even after the failure of the GPS and for parking the car in a safe location nearby.*
The Vehicle uses a Lidar from which the point cloud of obstacles surrounding the vehicle is obtained. Using several filters, only the static obstacles like the Traffic signal post are clipped. The vehicle is assumed to be at the stop line whose global coordinates are known. From the filtered point cloud data, the coordinates of the static reference points with respect to the vehicle are obtained. By using Parallelogram Law of Vectors, the global coordinates of the reference points are calculated. Then as the vehicle moves, the global coordinates of the vehicle is calculated from the corresponding local coordinates of the reference points with respect to the car. The Autonomous vehicle is modeled and imported in simulation software called Gazebo which runs alongside Robot Operating System. The Autonomous vehicle is mounted with a Velodyne HDL-32E Lidar, which is also modeled and imported in Gazebo. Point Cloud Library provides various filters required for processing the point cloud data. The Programming language used is python. Robot Operating System bridges between Gazebo and the algorithm developed.
.

Keywords: *Automatic Vehicle Detection System, Cloud Computing, Gazebo, Robot operating System* **.**

I. INTRODUCTION

An Autonomous car is a vehicle that is capable of sensing its environment and navigating without human input. Autonomous cars use a variety of techniques to detect the surroundings, such as radar, lidar, GPS, odometry and computer vision. There is a possibility of GPS failure, consequently, the car may lose its track of its position leading to an accident. My project is aimed at stimulating the motion of the car during GPS failure in a simulation environment called Gazebo. The technical problems identified are localization of the vehicle is not possible without the use of a GPS and Fail-safe system is required to localize the vehicle under all circumstances.

The Car is mounted with a camera. Once the GPS failures, a picture of the surrounding is taken and a known landmark is identified using various image processing techniques and the global coordinates embedded with the landmark is retrieved, thus the car is approximately localized.

Cameras cannot be used at night. Different lighting condition affects the identification of the landmark. The car can only be localized at the instant of GPS failure and cannot be moved to another specific location. Landmarks may not be available at the place of GPS failure.

The IEEE paper “Simultaneous Localization and Mapping” published by T. Bailey discusses the recursive Bayesian formulation of the simultaneous localization and mapping (SLAM) problem in which probability distributions or estimates of absolute or relative locations of landmarks and vehicle pose are obtained[7]. The paper focuses on three key areas: computational complexity; data association; and environment representation.[6]

The paper "Simultaneous map building and localization for an autonomous mobile robot" published by J.J. Leonard discusses a significant open problem in mobile robotics: simultaneous map building and localization, which the authors define as long-term globally referenced position estimation without a priori information[4]. This problem is difficult because of the following paradox: to move precisely, a mobile robot must have an accurate environment map; however, to build an accurate map, the mobile robot's sensing locations must be known precisely. In this way, simultaneous map building and localization can be seen to present a question of 'which came first, the chicken or the egg?' when using ultrasonic sensing, to overcome this issue the authors equip the vehicle with multiple servo-mounted sonar sensors, to provide a means in which a subset of environment features can be precisely learned from the robot's initial location and subsequently tracked to provide precise positioning.

The paper "A solution to the simultaneous localization and map building (SLAM) problem" published by M.W.M.G. Dissanayake provides a solution to the SLAM problem[5]. The simultaneous localization and map building (SLAM) problem asks if it is possible for an autonomous vehicle to start in an unknown location in an unknown environment and then to incrementally build a map of this environment while simultaneously using this map to compute absolute vehicle location. Starting from estimation-theoretic foundations of this problem, the paper proves that a solution to the SLAM problem is indeed possible. The underlying structure of the SLAM problem is first elucidated. A proof that the estimated map converges monotonically to a relative map with zero uncertainty is then developed. It is then shown that the absolute accuracy of the map and the vehicle location reach a lower bound defined only by the initial vehicle uncertainty. Together, these results show that it is possible for an autonomous vehicle to start in an unknown location in an unknown environment and, using relative observations only, incrementally build a perfect map of the world and to compute simultaneously a bounded estimate of vehicle location. The paper also describes a substantial implementation of the SLAM algorithm on a vehicle operating in an outdoor environment using millimeter-wave radar to provide relative map observations. This implementation is used to demonstrate how some key issues such as map management and data association can be handled in a practical environment. The results obtained are cross-compared with absolute locations of the map landmarks obtained by surveying. In conclusion, the paper discusses a number of key issues raised by the solution to the SLAM problem including suboptimal map-building algorithms and map management.

The IEEE paper "Natural Landmark-based autonomous vehicle navigation" published by R. Madhavan provides some information on localization as well as navigation of autonomous vehicle using natural landmarks. This article describes a natural landmark navigation algorithm for autonomous vehicles operating in relatively unstructured environments. The algorithm employs points of maximum curvature, extracted from laser scan data, as point landmarks in an extended Kalman filter[8]. A curvature scale space algorithm is developed to locate points of maximum curvature. The location of these points is invariant to view-point and sensor resolution. They can therefore be used as stable and reliable landmarks in a localization algorithm. This information is then fused with odometric information to provide localization information for an outdoor vehicle. The method described is invariant to the size and orientation of the range images under consideration (with respect to rotation and translation), is robust to noise, and can reliably detect and localize naturally occurring landmarks in the operating environment[9]. The method developed is generally applicable to a range of unstructured environments and may be used with different types of sensors.

II. ALGORITHM

First, the car is assumed to be at the stop line after the GPS failure. The global coordinates of the stop line is known from the map. The filtered point cloud data gives the position of the landmark with respect to the car. The goal is to calculate the global coordinates of the landmark from the global coordinates of the stop line and the coordinates of the landmark with respect to the car. The world frame and the car frame may be at different orientation. For the calculation, both of these frames should be of the same orientation, to achieve this the car frame is rotated which gives the coordinates of the landmark with respect to the car in the orientation of the world frame (Fig.1).

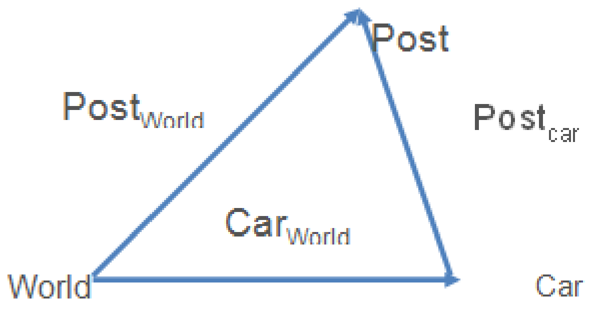
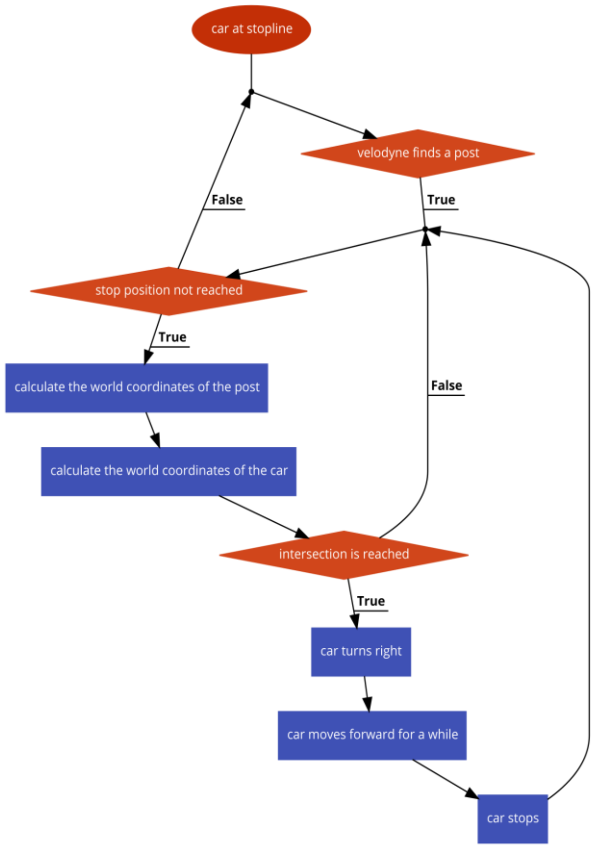


Fig.1: Calculation of Global Coordinates

Car_{World} is the global coordinates of the car, $Post_{car}$ is the coordinates of the post (landmark) with respect to the car and $Post_{world}$ is the coordinates of the post with respect to the world frame.

Car_{world}, $Post_{car}$ are known. $Post_{world}$ is calculated by subtracting car_{post} from $car_{world.}$ Then when the car moves, new coordinates of the post with respect to the car is obtained. From the global coordinates of the post which was calculated earlier and the coordinates of the post with respect to the car, the global coordinates of the car is calculated. Car_{world} is calculated by subtracting $Post_{car}$ from the $Post_{world}$.
The flowchart below gives the information about how the algorithm works(Fig.2).

Fig.2: Flow chart of the algorithm
First the car is assumed to be at the stop line whose


car at stopline
velodyne finds a post
False
True
stop position not reached
True
calculate the world coordinates of the post
calculate the world coordinates of the car
False
intersection is reached
True
car turns right
car moves forward for a while
car stops

coordinates are already known. From the velodyne point cloud data, the static reference points are obtained. Then the global coordinates of those reference points are calculated. So as the car moves forward, the global coordinates of the car is calculated continuously from the global coordinates of the reference points.
The car moves until the intersection whose global coordinates are known, then the car turn right and moves forward until a safe position is reached where the car stops.

III. MODELLING

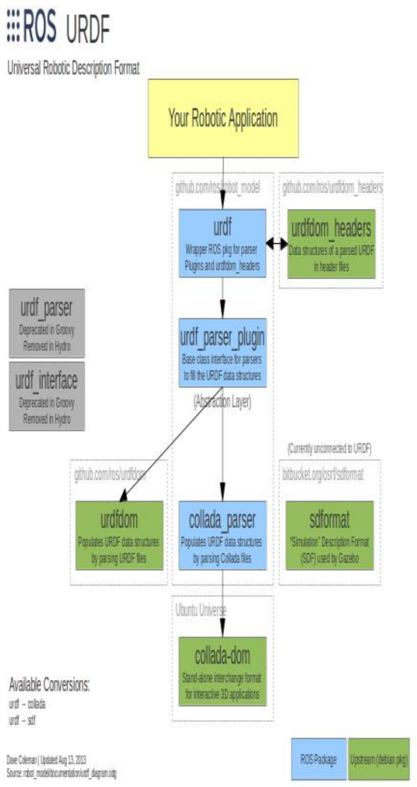

ROS URDF
Universal Robotic Description Format
Your Robotic Application
github.com/ros/robot_model
github.com/ros/urdfdom_headers
urdf
Wrapper ROS pkg for parser Plugins and urdfdom_headers
urdfdom_headers
Data structures of a parsed URDF in header files
urdf_parser
Deprecated in Groovy Removed in Hydro
urdf_interface
Deprecated in Groovy Removed in Hydro
urdf_parser_plugin
Base class interface for parsers to fill the URDF data structures
(Abstraction Layer)
(Currently unconnected to URDF)
github.com/ros/urdfdom
bitbucket.org/osrf/sdformat
urdfdom
Populates URDF data structures by parsing URDF files
collada_parser
Populates URDF data structures by parsing Collada files
sdformat
"Simulation" Description Format (SDF) used by Gazebo
Ubuntu Universe
collada-dom
Stand-alone interchange format for interactive 3D applications
Available Conversions:
urdf – collada
urdf – sdf
Dave Coleman | Updated Aug 13, 2013
Source: robot_model/documentation/urdf_diagram.odg
ROS Package
Upstream (debian pkg)

Fig.3 The Universal Robotic Description Format

The Universal Robotic Description Format (URDF) is an XML file format used in ROS to describe all elements of a robot[1-2]. To use a URDF file in Gazebo, some additional simulation-specific tags must be added to work properly with Gazebo[3].While URDFs are a useful and standardized format in ROS, they are lacking many features and have not been updated to deal with the evolving needs of robotics[10]. URDF can only specify the kinematic and dynamic properties of a single robot in isolation. URDF can not specify the pose of the robot itself within a world. It is also not a universal description format since it cannot specify joint loops (parallel linkages), and it lacks friction and other properties. Additionally, it cannot specify things that are not robots, such as lights, heightmaps, etc. The Architecture of URDF is represented in Fig.3
On the implementation side, the URDF syntax breaks proper formatting with heavy use of XML attributes, which in turn makes URDF more inflexible. There is also no mechanism for backward compatibility.

The main limitation at this point is that only tree structures can be represented, ruling out all parallel robots. Also, the specification assumes the robot consists of rigid links connected by joints; flexible elements are not supported.

To overcome the issues with the URDF, a new format called the Simulation Description Format (SDF) was created for use in Gazebo to solve the shortcomings of URDF. SDF is a complete description for everything from the world level down to the robot level. It is scalable, and makes it easy to add and modify elements. The SDF format is itself described using XML, which facilitates a simple upgrade tool to migrate old versions to new versions. It is also self-descriptive.

3.1 CADILLAC SRX

The Cadillac SRX is the Autonomous vehicle being developed at Carnegie Mellon University, Pittsburgh. The same car is to be imported into the simulation environment. The specification of the car such as the height, width, length, wheelbase, the radius of the wheel, turning radius is set up using a Unified Robot Description Format File (URDF). A 3D model of the Cadillac SRX (Fig.4) is also imported as a mesh in the URDF. Various plugins such as odometry, sonar are also added. Then the URDF is spawned in Gazebo.

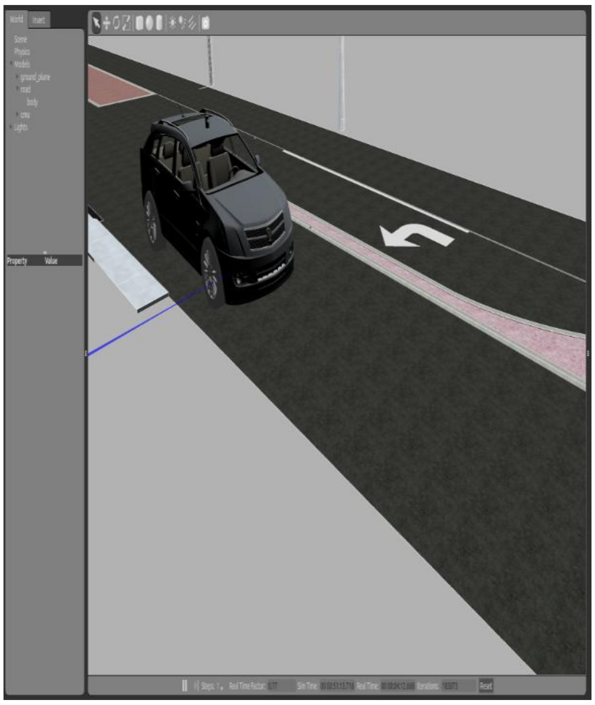


Fig. 4: Cadillac SRX Model in Gazebo

Velodyne HDL-32E is a 32 beam Lidar sensor. A model of Velodyne (Fig.4.3) is created using a Simulation Description Format file (SDF). The plugins needed for the rotation of the Lidar is also coded and the same is used in the Car’s URDF filre.

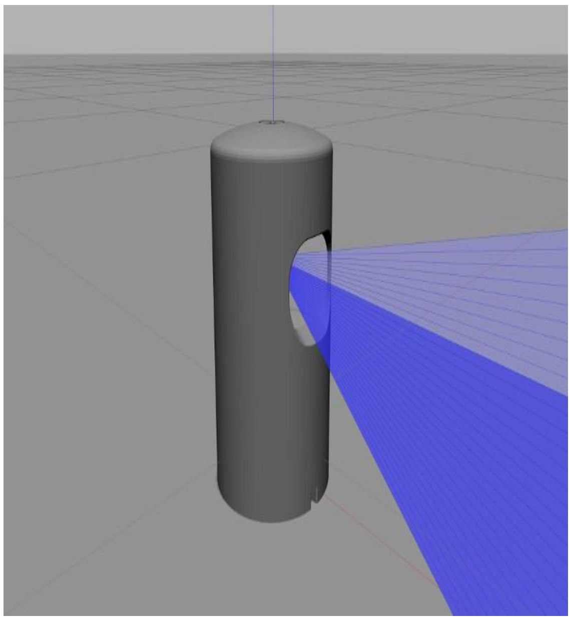


Fig. 5: Velodyne HDL-32E Model in Gazebo

A single Lidar is mounted on top of the car. The Point Cloud data is plublished on a topic for further processing.
A scenario of a road with a T- intersection (Fig.6) has been modeled using Sketchup and the same is imported as mesh in Gazebo. There are several static landmarks such as traffic lights, Bus stop sign and a stop line.

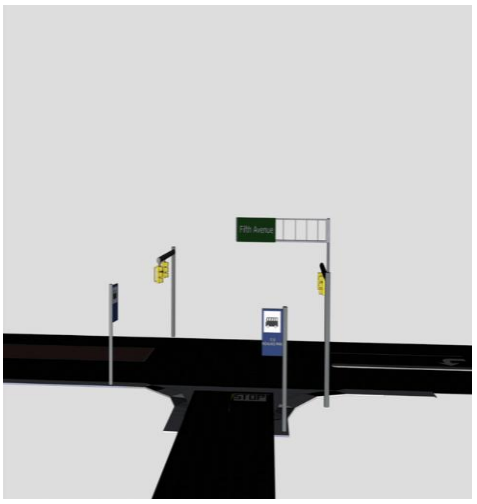
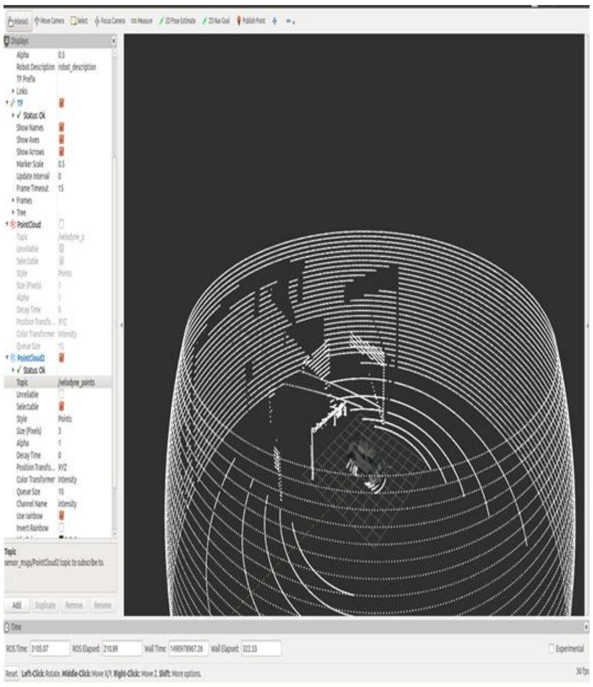


Fig.6: Intersection of Road in Gazebo

IV. RESULTS

The Velodyne mounted on top of the car scans the environment in Gazebo surrounding the car. The laser scan data is then converged as point clouds using Point Cloud library. The Point cloud data can be visualized in Rviz (Fig.7).



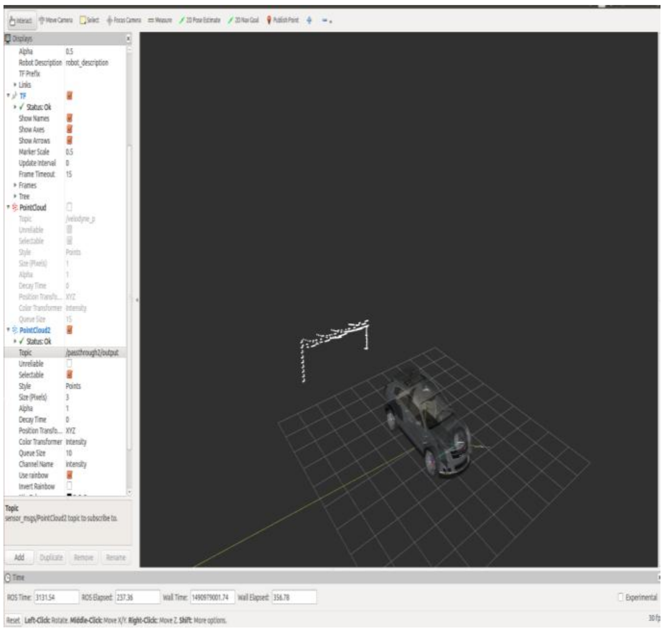
```
Displays
Alpha                 0.5
Robot Description     robot_description
TF Prefix
Links
TF                    ✓
  Status: Ok
  Show Names          ✓
  Show Axes           ✓
  Show Arrows         ✓
  Marker Scale        0.5
  Update Interval     0
  Frame Timeout       15
  Frames
  Tree
PointCloud
  Topic               /velodyne_p
  Unreliable
  Selectable          ✓
  Style               Points
  Size (Pixels)       1
  Alpha               1
  Decay Time          0
  Position Transfo... XYZ
  Color Transformer   Intensity
  Queue Size          15
PointCloud2           ✓
  Status: Ok
  Topic               /velodyne_points
  Unreliable
  Selectable          ✓
  Style               Points
  Size (Pixels)       3
  Alpha               1
  Decay Time          0
  Position Transfo... XYZ
  Color Transformer   Intensity
  Queue Size          10
  Channel Name        intensity
  Use rainbow         ✓
  Invert Rainbow

Topic
sensor_msgs/PointCloud2 topic to subscribe to.

ROS Time: 3105.07   ROS Elapsed: 210.89   Wall Time: 1490978967.26   Wall Elapsed: 322.33
```

Fig. 7: Point Cloud Visualization in Rviz

After applying a Pass-through filter to the obtained point cloud data, only the static landmarks such as the traffic light post gets clipped as shown in Fig.8.



```
PointCloud2           ✓
  Status: Ok
  Topic               /passthrough/output
  Style               Points
  Size (Pixels)       3

ROS Time: 3131.54   ROS Elapsed: 237.36   Wall Time: 1490979001.74   Wall Elapsed: 356.78
```

Fig. 8: Point Cloud after Applying Filter

CONCLUSION

The simulation of the proposed algorithm indicated that the algorithm can be successfully implemented. Accidents are reduced. Future work includes implementation of the algorithm for the motion of the car on different road configurations and also with different levels of traffic.

REFERENCES

[1] Lentin Joseph, Mastering ROS for Robotics Programming: Design, build and simulate complex robots using Robot Operating System and master its out-of-the-box functionalities. 2015

[2] Morgan Quigley, Brian Gerkey, William D. Smart, Programming Robots with ROS: A Practical Introduction to the Robot Operating System. 2015

[3] Anis Koubaa (eds.), Robot Operating System (ROS): The Complete Reference(1). 2016

[4] Jason M. O'Kane, A Gentle Introduction to ROS. 2015

[5] S. Argamon-Engelson, "Using image signatures for place recognition", Pattern Recognit. Lett., (19)pp. 941-951, 1998.

[6] T. Bailey, Mobile robot localisation and mapping in extensive outdoor environments, Univ. Sydney, 2002.

[7] T. Bailey, "Constrained initialisation for bearing-only SLAM", Proc. IEEE Int. Conf. Robotics Automation, pp. 1966-1971, 2003.

[8] Alexander Cunningham, Kai M. Wurm, Wolfram Burgard, Frank Dellaert, "Fully distributed scalable smoothing and mapping with robust multi-robot data association", Robotics and Automation (ICRA) 2012 IEEE International Conference on, pp. 1093-1100, 2012.

[9] Maxime Lhuillier, "Fusion of GPS and structure-from-motion using constrained bundle adjustments", Computer Vision and Pattern Recognition (CVPR) 2011 IEEE Conference on, pp. 3025-3032, 2011.

[10] D. R. Wong, M.P. Hayes, A. Bainbridge-Smith, "IMU-aided SURF feature matching for relative pose estimation", Image and Vision Computing New Zealand (IVCNZ) 2010 25th International Conference of, pp. 1-6,2010.