Write a 2000-word methodology for conducting a simulation of mobile robot in ROS2 using SLAM and Nav2 toolbox. Use the following blue print of the entire process.

1. Purpose of mobile robot: To plough the drying cashews on the ground continuously 2 to 3 times a day to reduce moisture content.
2. Design of mobile robot: Physical dimensions are modelled by using Solidworks software. CAD model is developed.
3. Exporting the CAD design of mobile robot: URDF file of cad model is exported to ROS2 workspace and importing them in the package.
4. downloading and installing necessary plugins for sensors
5. writing launch files
6. downloading the slam toolbox and nav2
7. Environment creation: Environment is created using Blendr. A typical village ground is modelled. Trees and bushes are modelled as obstacles. Then building the world and generating the map.
8. Then moving the bot from the set point to generate a goal point.

explain point 6 with more detail , use following information if necessary

Nav2 (Navigation2) is a navigation stack in ROS2 that provides a set of navigation-related functionalities, such as localization, path planning, and obstacle avoidance. Nav2 is designed to work with various types of robots, including ground, aerial, and underwater vehicles.

Nav2 is separate from SLAM (Simultaneous Localization and Mapping), which is a process of building a map of an unknown environment while simultaneously localizing the robot within that map. While Nav2 uses some SLAM-related functionalities, such as localization, it is primarily focused on navigation and path planning, rather than map building.

Nav2 provides a modular architecture that allows for the integration of various localization and mapping algorithms, including SLAM algorithms. For example, Nav2 can use a pre-built map generated by a SLAM algorithm for localization and path planning.

In summary, while Nav2 can use SLAM-related functionalities, it is primarily focused on navigation and path planning, and therefore is a separate component from SLAM.

Algorithms:

The SLAM toolbox in ROS2 provides several algorithms for simultaneous localization and mapping (SLAM), which is the process of building a map of an unknown environment while simultaneously localizing the robot within that map. The specific algorithm used by the SLAM toolbox in ROS2 depends on the selected package and configuration.

For example, the GMapping package in ROS2 uses a grid-based Rao-Blackwellized particle filter to perform SLAM. This algorithm combines a particle filter with a grid-based map representation to estimate the robot's position and orientation while building a map of the environment.

Another popular algorithm used in ROS2 SLAM is Hector SLAM, which is a scan matching-based method that uses a 2D laser scanner to estimate the robot's position and orientation. It builds a map of the environment using a combination of occupancy grids and distance transforms.

In addition to these algorithms, ROS2 provides other SLAM packages such as Cartographer, which is a real-time 2D and 3D SLAM system that uses simultaneous localization and mapping to build maps of indoor environments.

Research paper algos

The basis for the EKF-SLAM method is to describe the

vehicle motion in the form

P(xk | xk−1, uk) ⇐⇒ xk = f(xk−1, uk) + wk, (6)

where f(·) models vehicle kinematics and where wk are

additive, zero mean uncorrelated Gaussian motion disturbances with covariance Qk. The observation model is described in the form

P(zk | xk, m) ⇐⇒ z(k) = h(xk, m) + vk, (7)

where h(·) describes the geometry of the observation and

where vk are additive, zero mean uncorrelated Gaussian

observation errors with covariance Rk

With these definitions the standard EKF method [31],

[14] can be applied to compute the mean

·

xˆk|k

mˆ k

¸

= E·

xk

m

| Z0:k

¸

,

and covariance

Pk|k =

·

Pxx Pxm

PT

xm Pmm ¸

k|k

= E"µ

xk − xˆk

m − mˆ k

¶ µ xk − xˆk

m − mˆ k

¶T

| Z0:k

#

of the joint posterior distribution P(xk, m | Z0:k, U0:k, x0)

from:

Time-update

xˆk|k−1 = f(xˆk−1|k−1, uk) (8)

Pxx,k|k−1 = ∇f Pxx,k−1|k−1∇f

T + Qk (9)

where ∇f is the Jacobian of f evaluated at the estimate

xˆk−1|k−1. There is generally no need to perform a timeupdate for stationary landmarks3

.

Observation-update

·

xˆk|k

mˆ k

¸

=

·

xˆk|k−1

mˆ k−1

¸

+Wk

£

z(k) − h(xˆk|k−1, mˆ k−1)

¤

(10)

Pk|k = Pk|k−1 − WkSkWT

k

(11)

where

Sk = ∇hPk|k−1∇h

T + Rk

Wk = Pk|k−1∇h

T S

−1

k

and where ∇h is the Jacobian of h evaluated at xˆk|k−1 and

mˆ k−1.

This EKF-SLAM solution is very well known and inherits

many of the same benefits and problems as the standard

EKF solutions to navigation or tracking problems. Four of

the key issues are briefly discussed here:

Convergence: In the EKF-SLAM problem, convergence

of the map is manifest in the monotonic convergence of the

determinant of the map covariance matrix Pmm,k, and all

land-mark pair sub-matrices, toward zero. The individual

land-mark variances converge toward a lower bound determined by initial uncertainties in robot position and observations. The typical convergence behaviour of landmark

location variances is shown in Figure 3 (from [14]).

Computational Effort: The observation update step requires that all landmarks and the joint covariance matrix

be updated every time an observation is made. Naively,

this means computation grows quadratically with the number of landmarks. There has been a great deal of work undertaken in developing efficient variants of the EKF-SLAM

solution and real-time implementations with many thousands of landmarks have been demonstrated [21], [29]. Efficient variants of the EKF-SLAM algorithm are discussed

in Part II of this tutorial.

Data Association: The standard formulation of the EKF-SLAM solution is especially fragile to incorrect association of observations to landmarks [35]. The ‘loop-closure’ problem, when a robot returns to re-observe landmarks after a large traverse, is especially difficult. The association problem is compounded in environments where landmarks are not simple points and indeed look different from different view-points. Current work in this area will be described in Part II of this tutorial.

Non-linearity: EKF-SLAM employs linearised models of non-linear motion and observation models and so inherits many caveats. Non-linearity can be a significant problem in EKF-SLAM and leads to inevitable, and sometimes dramatic, inconsistency in solutions [24]. Convergence and consistency can only be guaranteed in the linear case.

Rao backlisted

The FastSLAM algorithm, introduced by Montemerlo et al. [32], marked a fundamental conceptual shift in the design of recursive probabilistic SLAM. Previous efforts focused on improving the performance of EKF-SLAM, while retaining its essential linear Gaussian assumptions. FastSLAM with its basis on recursive Monte Carlo sampling, or particle filtering, was the first to directly represent the non-linear process model and non-Gaussian pose distribution.4 This approach was influenced by earlier probabilistic mapping experiments of Murphy [34] and Thrun [41]. The high dimensional state-space of the SLAM problem makes direct application of particle filters computationally infeasible. However, it is possible to reduce the sample-space by applying Rao-Blackwellisation (R-B),

whereby a joint state is partitioned according to the product rule P(x1, x2) = P(x2 | x1)P(x1) and, if P(x2 | x1) can be represented analytically, only P(x1) need be sampled x (i) 1 ∼ P(x1). The joint distribution, therefore, is represented by the set {x (i) 1 , P(x2 | x (i) 1 )} N i and statistics such as the marginal P(x2) ≈ 1 N X N i P(x2 | x (i) 1 )

can be obtained with greater accuracy than is possible by sampling over the joint space. The joint SLAM state may be factored into a vehicle component and a conditional map component. P(X0:k, m | Z0:k, U0:k, x0) = P(m | X0:k, Z0:k)P(X0:k | Z0:k, U0:k, x0).

Here the probability distribution is on the trajectory X0:k rather than the single pose xk because, when conditioned on the trajectory, the map landmarks become independent (see Figure 4). This is a key property of FastSLAM, and the reason for its speed; the map is represented as a set of independent Gaussians, with linear complexity, rather than a joint map covariance with quadratic complexity

A graphical model of the SLAM algorithm. If the history of pose states are known exactly then, since the observations are conditionally independent, the map states are also independent. For FastSLAM, each particle defines a different vehicle trajectory hypothesis

The essential structure of FastSLAM, then, is a RaoBlackwellised state, where the trajectory is represented by weighted samples and the map is computed analytically. Thus, the joint distribution, at time k, is represented by the set {w (i) k , X (i) 0:k , P(m | X (i) 0:k , Z0:k)} N i , where the map accompanying each particle is composed of independent Gaussian distributions P(m | X (i) 0:k , Z0:k) = QM j P(mj | X (i) 0:k , Z0:k). Recursive estimation is performed by particle filtering for the pose states, and the EKF for the map states. Updating the map, for a given trajectory particle X (i) 0:k , is trivial. Each observed landmark is processed individually as an EKF measurement update from a known pose (see Figure 5). Unobserved landmarks are unchanged. Propagating the pose particles, on the other hand, is more complex, as we discuss below

We forego giving a background on particle filters, except to say that it is derived from a recursive form of sampling known as sequential important sampling (SIS) [15], which actually samples from a joint state history, but “telescopes” the joint into a recursion via the product rule. P(x0, x1, . . . , xT | Z0:T ) = P(x0 | Z0:T )P(x1 | x0, Z0:T ). . . P(xT | X0:T −1, Z0:T ). At each time-step k, particles are drawn from a proposal distribution π(xk | X0:k−1, Z0:k), which approximates the true distribution P(xk | X0:k−1, Z0:T ), and the samples are given importance weights to compensate for any discrepancy. The approximation error grows with time (and inherent joint state-space), increasing the variation in sample weights, degrading statistical accuracy. A resampling step reinstates uniform weighting, but causes loss of historical particle information. This leads to a crucial property: SIS with resampling can produce reasonable statistics only for systems that “exponentially forget” their past [8] (i.e., systems whose process noise cause the state at time k to become increasingly independent of preceding states). The general form of a R-B particle filter for SLAM is as follows. We assume that, at time k − 1, the joint state is represented by {w (i) k−1 , X (i) 0:k−1 , P(m | X (i) 0:k−1 , Z0:k−1)} N i . 1. For each particle, compute a proposal distribution, conditioned on the specific particle history, and draw a sample from it x (i) k ∼ π(xk | X (i) 0:k−1 , Z0:k, uk). (13) This new sample is (implicitly) joined to the particle history X (i) 0:k 4 = n X (i) 0:k−1 , x (i) k o . 2. Weight samples according to the importance function w (i) k = w (i) k−1 P(zk | X (i) 0:k , Z0:k−1)P(x (i) k | x (i) k−1 , uk) π(x (i) k | X (i) 0:k−1 , Z0:k, uk) . (14) The numerator terms of this equation are the observation model and the motion model, respectively. The former differs from Equation 2 because R-B requires dependency on the map be marginalised away. P(zk | X0:k, Z0:k−1) = Z P(zk | xk, m)P(m | X0:k−1, Z0:k−1)dm (15) 3. If necessary,5 perform resampling. Resampling is accomplished by selecting particles, with replacement, from the set {X (i) 0:k } N i , including their associated maps, with probability of selection proportional to w (i) k . Selected particles are given uniform weight, w (i) k = 1 N . 4. For each particle, perform an EKF update on the observed landmarks as a simple mapping operation with known vehicle pose. The two versions of FastSLAM in the literature, FastSLAM 1.0 [32] and FastSLAM 2.0 [33], differ only in terms of the form of their proposal distribution (step 1) and, consequently in their importance weight (step 2). FastSLAM 2.0 is by far the more efficient solution. For FastSLAM 1.0, the proposal distribution is the motion model x (i) k ∼ P(xk | x (i) k−1 , uk) (16) Therefore, from Equation 14, the samples are weighted according to the marginalised observation model. w (i) k = w (i) k−1P(zk | X (i) 0:k , Z0:k−1) (17) For FastSLAM 2.0, the proposal distribution includes the current observation x (i) k ∼ P(xk | X (i) 0:k−1 , Z0:k, uk) where P(xk | X (i) 0:k−1 , Z0:k, uk) = 1 C P(zk | xk, X (i) 0:k−1 , Z0:k−1)P(xk | x (i) k−1 , uk)(19) and C is a normalising constant. The importance weight according to Equation 14 is w (i) k = w (i) k−1C. The advantage of FastSLAM 2.0 is that its proposal distribution is locally optimal [15]. That is, for each particle, it gives the smallest possible variance in importance weight w (i) k conditioned upon the available information, X (i) 0:k−1 , Z0:k and U0:k. Statistically, FastSLAM (1.0 and 2.0) suffers degeneration due to its inability to forget the past. Marginalising the map in Equation 15 introduces dependence on the pose and measurement history, and so, when resampling depletes this history, statistical accuracy is lost [2]. Nevertheless, empirical results of FastSLAM 2.0 in real outdoor environments [33] show that the algorithm is capable of generating an accurate map in practice

**Ceres solver**

* **Code Quality** - Ceres Solver has been used in production at Google for more than four years now. It is clean, extensively tested and well documented code that is actively developed and supported.
* **Modeling API** - It is rarely the case that one starts with the exact and complete formulation of the problem that one is trying to solve. Ceres’s modeling API has been designed so that the user can easily build and modify the objective function, one term at a time. And to do so without worrying about how the solver is going to deal with the resulting changes in the sparsity/structure of the underlying problem.
  + **Derivatives** Supplying derivatives is perhaps the most tedious and error prone part of using an optimization library. Ceres ships with [automatic](http://en.wikipedia.org/wiki/Automatic_differentiation) and [numeric](http://en.wikipedia.org/wiki/Numerical_differentiation) differentiation. So you never have to compute derivatives by hand (unless you really want to). Not only this, Ceres allows you to mix automatic, numeric and analytical derivatives in any combination that you want.
  + **Robust Loss Functions** Most non-linear least squares problems involve data. If there is data, there will be outliers. Ceres allows the user to *shape* their residuals using a **[LossFunction](http://ceres-solver.org/nnls_modeling.html" \l "_CPPv4N5ceres12LossFunctionE" \o "ceres::LossFunction)** to reduce the influence of outliers.
  + **Manifolds** In many cases, some parameters lie on a manifold other than Euclidean space, e.g., rotation matrices. In such cases, the user can specify the geometry of the local tangent space by specifying a [**Manifold**](http://ceres-solver.org/nnls_modeling.html#_CPPv4N5ceres8ManifoldE) object.
* **Solver Choice** Depending on the size, sparsity structure, time & memory budgets, and solution quality requirements, different optimization algorithms will suit different needs. To this end, Ceres Solver comes with a variety of optimization algorithms:
  + **Trust Region Solvers** - Ceres supports Levenberg-Marquardt, Powell’s Dogleg, and Subspace dogleg methods. The key computational cost in all of these methods is the solution of a linear system. To this end Ceres ships with a variety of linear solvers - dense QR and dense Cholesky factorization (using [Eigen](http://eigen.tuxfamily.org/), [LAPACK](http://www.netlib.org/lapack/) or [CUDA](https://developer.nvidia.com/cuda-toolkit)) for dense problems, sparse Cholesky factorization ([SuiteSparse](http://www.cise.ufl.edu/research/sparse/SuiteSparse/), [`Accelerate`\_](http://ceres-solver.org/features.html#id1), [Eigen](http://eigen.tuxfamily.org/)) for large sparse problems, custom Schur complement based dense, sparse, and iterative linear solvers for [bundle adjustment](http://en.wikipedia.org/wiki/Bundle_adjustment) problems.
  + **Line Search Solvers** - When the problem size is so large that storing and factoring the Jacobian is not feasible or a low accuracy solution is required cheaply, Ceres offers a number of line search based algorithms. This includes a number of variants of Non-linear Conjugate Gradients, BFGS and LBFGS.
* **Speed** - Ceres Solver has been extensively optimized, with C++ templating, hand written linear algebra routines and OpenMP or modern C++ threads based multithreading of the Jacobian evaluation and the linear solvers.
* **GPU Acceleration** If your system supports [CUDA](https://developer.nvidia.com/cuda-toolkit) then Ceres Solver can use the Nvidia GPU on your system to speed up the solver.
* **Solution Quality** Ceres is the [best performing](https://groups.google.com/forum/#!topic/ceres-solver/UcicgMPgbXw) solver on the NIST problem set used by Mondragon and Borchers for benchmarking non-linear least squares solvers.
* **Covariance estimation** - Evaluate the sensitivity/uncertainty of the solution by evaluating all or part of the covariance matrix. Ceres is one of the few solvers that allows you to do this analysis at scale.
* **Community** Since its release as an open source software, Ceres has developed an active developer community that contributes new features, bug fixes and support.
* **Portability** - Runs on *Linux*, *Windows*, *Mac OS X*, *Android* *and iOS*.
* **BSD Licensed** The BSD license offers the flexibility to ship your application

***Fast SLAM***

The Fast SLAM algorithm is a probabilistic simultaneous localization and mapping (SLAM) algorithm that uses recursive Monte Carlo sampling, also known as particle filtering. It was first introduced by Montemerlo et al. in a 2002 paper. Unlike earlier SLAM algorithms, which were based on the Extended Kalman Filter (EKF) and made linear Gaussian assumptions, Fast SLAM directly represents the non-linear process model and non-Gaussian pose distribution.

Fast SLAM uses Rao-Blackwellisation to reduce the sample space, which involves partitioning a joint state according to the product rule and sampling only the parts that cannot be represented analytically. The joint SLAM state can be factored into a vehicle component and a conditional map component, which allows the map landmarks to become independent when conditioned on the trajectory. This means the map can be represented as a set of independent Gaussians, rather than a joint map covariance, making it faster and more efficient.

Fast SLAM uses a Rao Blackwellised state, where the trajectory is represented by weighted samples and the map is computed analytically. The joint distribution, at time k, is represented by the set {w (i) k , X (i) 0:k , P(m | X (i) 0:k , Z0:k)} N i , where the map accompanying each particle is composed of independent Gaussian distributions. Recursive estimation is performed by particle filtering for the pose states, and the EKF for the map states. Updating the map, for a given trajectory particle, is trivial, while propagating the pose particles is more complex.

Fast SLAM uses particle filtering, which involves drawing particles from a proposal distribution and giving them importance weights to compensate for any discrepancy. The approximation error grows with time, increasing the variation in sample weights and degrading statistical accuracy. A resampling step reinstates uniform weighting but causes loss of historical particle information. Fast SLAM with resampling can only produce reasonable statistics for systems that "exponentially forget" their past, or systems whose process noise causes the state at time k to become increasingly independent of preceding states.