Evaluation of Simultaneous Localization and Mapping and Path Planning Algorithms of a Mobile Vehicle

Ramachandran, A.¹, and Raol, J. R.²

¹Professor, Dept. of Electronics & Instrumentation Engg., MSRIT and Research Scholar, VTU, Belgaum; India.
²Professor Emeritus, MSRIT, Bangalore; India

ABSTRACT:

Two applications in the context of robotics are considered: i) the problem of simultaneous localization and mapping (SLAM) for a mobile vehicle, especially for a mobile robot using H-infinity filtering, and ii) the robot path planning (PP) using several algorithms; including two soft computing approaches to path planning. After presenting the results of SLAM using MATLAB, a brief review of several PP algorithms and their results is covered. The latter are analysed in a composite manner with a simple metric, perhaps not done earlier, and this suggests that the bidirectional rapidly expanding random tree PP algorithm and the fuzzy logic based approaches are quite preferable to other competing PP algorithms.

KEYWORDS—Localization; mapping; land marks; way points; mobile vehicle; H-infinity filter; robot path planning; A* algorithm, potential field, rapidly exploring random tree, probabilistic road map, fuzzy logic; genetic algorithm.

I. INTRODUCTION

A mobile vehicle or a robot uses the measurements that are made relative to the locations to the landmarks in the vehicle’s environment, which is the one that the mobile vehicle is going to traverse through its planned journey; this is the problem of SLAM, in which we estimate the pose (robot’s location/its position coordinates) and the map (the coordinates of the landmarks-LMs) for a mobile vehicle at the same time. This problem is generally involved because: a) a map (its coordinates) is needed to localize (to pose) the vehicle, and ii) robot’s pose (its estimate) is needed to build a map; and from the system and control theory point of view this problem of SLAM calls for the joint state and parameter estimation: the trajectory of the vehicle’s platform and the location of available landmarks (map building) should be simultaneously estimated from a given set of measurements. The SLAM is studied in [1-5], however, for the present work it is studied in the H-infinity (HI) framework that gives robust filtering/estimation algorithms, and it is a novel application of the HI filter the SLAM. A solution to the joint localization and mapping problem requires a joint state composed of the vehicle pose, i.e. its position, and every landmark position, to be updated following each landmark measurement, often these measurements are noisy. This often requires using a state vector with large dimension (if the number of landmarks maintained in the map is very large) [4]. The solution to the SLAM problem would provide a means to make any vehicle and mobile robots truly autonomous, it is also useful for unmanned aerial vehicles (UAVs), micro-, mini-air vehicles (MICAV/MAVs) and many other types of autonomous vehicles ranging from indoor, hospital robots to outdoor robots/field robots, underwater vehicles (UWVs), unmanned ground vehicles (UGVs) and many airborne systems. The measurements are used by the vehicle’s (on-board) sensors and processed by the on-board computer to estimate the vehicle’s states and the landmarks locations. The state vector can contain the vehicle’s position, velocity and acceleration in any or more directions: x-, y-, and z - axes. In addition, it can contain the vehicle’s orientations (e.g. heading angle). The joint state estimator uses three mathematical models: a) the vehicle model, b) landmarks model, and c) the sensor’s model. The vehicle’s mathematical model describes the kinematics and dynamics of the vehicle and the sensors model relates the measurements to the state vector, the robot’s path-states to be estimated. The mathematical models can be considered to be linear for the sake of the simplicity of the exposition and to clearly understand the problem of estimation in SLAM, however, in reality the mathematical model for the vehicle is nonlinear one. Traditionally the SLAM is studied in the so called well known Kalman filtering (KF) setting, mainly the extended Kalman filter (EKF), because the SLAM model is a nonlinear one. In this paper, we study the SLAM problem in HI fame work that by definition is to provide robust filtering algorithms, and use the linearization approach for implementation in MATLAB. Hence, the main idea in SLAM is to place the vehicle at an unknown location in an unknown environment, and then the vehicle incrementally (with SLAM algorithm in its on-board computer) builds a consistent map of its environment while at the same time determining its location within this map. Figure 1 shows the SLAM problem geometry and it would become clearer with the study of the legends in the side-box.
Fig. 1 SLAM configuration (LMs dispositions); legends in the right side box.

Path planning is a task of accepting many input parameters for a robot to go from the current location to the desired location, and is very closely related application to the problem of SLAM; however, there are several approaches to robot PP algorithms, and we briefly review these algorithms and their associated results in a composite way. We also study two soft computing paradigms of fuzzy logic (FL), and genetic algorithm (GA) for robot PP: i) the GA would provide a potential solution to the PP problem, and it is used for optimal path finding in a space with obstacles; and ii) application of FL to address the local PP to escape from local minimum during goal-oriented robot navigation in unknown environment can be considered. Our closer study of these several algorithms’ results in a composite way suggests us to use a simple metric for performance acceptance and the minimum of which would point to the best PP algorithm amongst the band of several competing ones.

II. II SLAM

A mobile vehicle/robot moves or ‘walks’ through its surrounding taking relative measurements of a number of unknown LMs using sensors located on the vehicle, say at time $k$; then the important quantities for this case are defined as: i) the state vector of the location (x, y, and z positions) and orientation angles of the vehicle (say heading), ii) the control vector $u$, applied at time $k-1$ to move the vehicle to a state at time $k$, meaning the vehicle’s state will alter from $x(k-1)$ to $x(k)$, iii) a vector that describes the location of the i-th LM, the true location of which is, presently assumed time invariant, and iv) a set of measurements from the vehicle’s sensors about the location (relative measurement with respect to the vehicle location) of the i-th LM at time index $k$. The map-building depends on the poses of the vehicle during data acquisition: if the poses are known correctly, mapping is easy, and if the mapping is done accurately, the vehicle poses are accurate; so, the SLAM is a boot-strap or joint state/parameter estimation problem. The vehicle’s mathematical model is given as

$$x_v(k + 1) = f_v(x_v(k), u_v(k), k) + w_v(k)$$

(1)

In (1), $u$ is an input control vector (with velocity inputs and steering angles useful for actual motion planning problem), and $x$ is the (extended/expanded) state vector of the model (with vehicles’ states: position and velocity); ‘$f$’ is a vector-valued nonlinear function that models the mobility, kinematics and dynamics of the vehicle, and orientation dynamics. The un-modelled behaviour is captured in process noise $w$ which is assumed to be zero mean white Gaussian noise with covariance matrix $Q$. The LMs’ locations are considered fixed with some uncertainty (or sometimes without any uncertainty) and are represented as a point model. Here, we consider a simple LM model with two parameters with respect to some global reference coordinate frame. The LM model is time invariant, and i-th point LM is defined as

$$p_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}$$

(2)

The LM is considered stationary and the model is given as

$$p_i(k + 1) = p_i(k) = p_i$$

(3)
In the LM model no additive uncertainty is considered, since the LM location is assumed to be known precisely, however in actual practice this would not be so, and any such uncertainty can be easily incorporated into the model. The measurement model is given as

\[ z_i(k+1) = H_i(x_i(k+1), p_i, k+1) + v_i(k+1) \]  

(4)

In (4), the \( z \) is the observables’ vector of the LM location relative to the location of the vehicle, \( H \) is the sensor model, and \( v(.) \) is the zero mean white Gaussian measurement noise sequence with covariance matrix \( R \). For obtaining SLAM results by HI filtering algorithm, the linearization is used to compute the covariance propagation, data updates and filter gains.

2.1 H-INFINITY FILTER based SLAM

Traditionally, Kalman filtering (KF) algorithm and its many variants have been used for SLAM, these belong to the statistical/probability-based approaches; however, we consider the SLAM in the domain of joint state/parameter estimation using H-\( \infty \) (H infinity/HI) filtering approach that is based on the so called H-\( \infty \) norm. The numerical value of this norm has to be below a certain bound/threshold to obtain robust filtering algorithm. Hence, we present a relatively novel solution to the SLAM problem in utilizing the HI filtering. In the KF approach, the basic signal processing system is a state space model driven by a white noise process with known statistical properties (Q), and the measurement signals are assumed to be corrupted with the white noise of known (or assumed known) statistical properties (R covariance matrix). The aim in the KF is to minimize the variance of the output state estimation error; whereas the H-\( \infty \) filtering problem differs in mainly two respects [6]: i) the white noise is replaced by an unknown deterministic disturbance of a finite energy, and ii) a pre-specified positive real number, gamma (a scalar) is defined; then the goal of the HI filter is to ensure that the energy gain from the input disturbances to the output estimation error is less than the square of gamma; this number being called as a threshold for the magnitude of the transfer function (TF) between output estimation error variance (energy) and all the input disturbance energies (variances). From the point-of-view-of robustness we see that the H-\( \infty \) concept would yield a robust filtering algorithm if not the optimal one, because the maximum gain of the TF is kept below/under a bound. Hence, in this case the gamma turns out to be a tuning parameter for the HI filter. In many practical applications robustness aspect could be more important than mere optimality requirement. The H-\( \infty \) norm involves RMS (root-mean-square) value of a signal, i.e. a measure or a metric of a signal that reflects eventual average size of RMS value. The H-\( \infty \) norm [6] used is given as:

\[
\sum_{i=0}^{N} \sum_{k=0}^{N} [\hat{x}(k) - x(k)]^T H_{i}^T (\hat{x}(k) - x(k))
\]

\[
(\hat{x}_0 - x_0)^T P_{0} (\hat{x}_0 - x_0) + \sum_{k=0}^{N} [w(k)]^T [w(k)] + \sum_{k=0}^{N} [v_i(k)]^T [v_i(k)]
\]

(5)

It can be readily seen from (5) that the input to the filter consists of energies (represented as variances in the denominator) due to the errors in: a) the initial condition (initial state error), b) the state disturbance and c) the measurement disturbance; and the output energy of the filter (the numerator) is due to the error in the estimated states. Basically this ratio, the H-\( \infty \) norm, should be less than square of gamma, which is considered as an upper bound on the maximum energy gain from the input to the output, i.e. the worst case. We emphasize here, that no statistical assumptions on the noise processes are required to be made; so, by a class, and (5), the H-\( \infty \) filters are deterministic filters where the robustness is emphasized rather than the randomness of the signals and the optimality. It has been recently shown that H-\( \infty \) estimation/control problems related to risk-sensitive estimation and adaptive filtering are studied in a simple and unified manner in the indefinite metric space called a Krein space [7, 8]. In the conventional H-2 (Hilbert/Hardy space) framework, on which the celebrated KF is based [9], the unknown state vector and the additive disturbance (noises) are assumed to be stochastic variables. Robust solutions to the filtering problems in the H-\( \infty \) space are found by minimizing expected prediction error energy, as represented by the H-infinity norm (5). The variables and signals in HI framework are known as generalized random variables and the covariance matrices are called Gramians, because these now relate to the generalized variables and not to the stochastic processes as in the case of KF/EKF.
2.2 MATHEMATICAL MODEL AND H-\(\infty\) a POSTERIORI FILTER

The linear dynamic model of a mobile vehicle and the HI filter equations are given as

\[
x(k+1) = \phi x(k) + Gw(k) \\
z(k) = Hx(k) + v(k)
\]

\[
P(k+1) = \phi P(k) + \phi^T G Q G^T \phi + R \\
K_i = P_i (k+1) H_i^T (I + H_i P_i (k+1) H_i^T)^{-1}
\]

\[
\hat{x}_i(k+1) = \phi \hat{x}_i(k) + K_i (z_i(k+1) - H_i \phi \hat{x}_i(k))
\]

In (6) we have, \(\phi\) as the coefficient matrix of the state dynamics model, \(G\) as the process noise gain matrix/vector, the sampling interval is indexed by discrete-time index \(k\), and the LM’s relative measurements are modelled as \(H\) (the measurement/sensor model). The process and measurement noises, \(w(k)\), and \(v(k)\) are considered as deterministic disturbances; with respective Gramians as \(Q\) and \(R\). The state estimates, \(x(.)\) are obtained by a posterior filter [7, 8], i.e. after the measurements are made and included in the filter; \(L\) denotes the linear combination of state-estimates (used in Krein space); \(P\) is the (covariance) Gramian, and \(K\) is the HI gain (like Kalman gain matrix/function). The second term in the parenthesis of the state estimate equation gives the residuals. The merit of the HI filter is that the specification of covariance matrices of the process noise and measurement noise as required in the case of KF is not required here. As is often the case these matrices are not really and accurately known in real-life situations. In such cases, perhaps, a very suitable approach would be to use any HI filter that is based on the assumptions of deterministic input disturbances and their finite energies.

2.3 SIMUALTION RESULTS for SLAM WITH HI FILTER

As the SLAM problem is a non-linear filtering problem we use linearization similar to that used in the EKF. Due to this linearization process the H-\(\infty\) filter is directly applicable to be incorporated into the SLAM structure, with some simple algebraic modifications. The vehicle states are: x-axis, y-axis and heading (phi) angle; defined as the vehicle-pose and specifies the vehicle model. It also includes the vehicle velocity, \(V\) (3 m/s). The mathematical model for the H infinity filter (in the MATLAB program) is given as [10]:

\[
xv = \{xv(1) + V*T*cos(G+xv(3,:)); \radian)

xv(2) + V*T*sin(G+xv(3,:));
pi_to_pi(xv(3) + V*T*sin(G)/VB)\}
\]

(7)

Here, \(xv\) is the vehicle pose (x,y, in meters, and phi in rad.), \(G\) is the steering angle, VB is the vehicle’s wheel base and \(T=0.025 \text{s}\). The Q matrix is chosen as diagonal with elements as \(Q = [0.09 0; 0 0.0027]\). Similarly, the measurement Gramian matrix is taken as \(R = [0.01 0; 0 0.0003]\). It should be noted here that this R is not the same as Re in equation (6). Also, it should be noted that the matrices Q and R can be used to generate usual random noise processes for adding to the states and measurements. In the real-life situations the data would be corrupted with such random noises. Still, the H-infinity filter can be used by assuming that these noise processes are generalized variables (or deterministic disturbances) and proceeding with the application of the HI filter. Then, the idea is to still obtain a good and robust solution to the estimation problem, if not the optimal one. The maximum range chosen is 30 meters. Maximum steering angle is 30 deg., and the maximum steering rate is 20 deg./sec. The disposition of a typical layout of the waypoints and landmarks is shown in Figure 2. The percentage fit (state-) errors for the x-axis, y-axis, and heading are given in Table I for various cases (of Q and R) and gamma values; we see that reasonably (very) low values of these errors have been achieved in the HI SLAM problem.
TABLE I

THE PERCENTAGE FIT ERRORS (PFE) FOR THE HI FILTER used in SLAM

<table>
<thead>
<tr>
<th>Case study</th>
<th>Values of noise variances (for simulated data)</th>
<th>Gamma in the H-infinity filter</th>
<th>PFITE in x-axis/position</th>
<th>PFITE in y-axis/position</th>
<th>PFITE in Heading</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>Nominal values of Q, R</td>
<td>1.05</td>
<td>2.5</td>
<td>2.06</td>
<td>1.8</td>
</tr>
<tr>
<td>ii</td>
<td>2 x the nominal values for each Q, R</td>
<td>1.3</td>
<td>2.8</td>
<td>3.0</td>
<td>1.4</td>
</tr>
<tr>
<td>iii</td>
<td>--Do-- (with different gamma)</td>
<td>1.5</td>
<td>5.8</td>
<td>5.9</td>
<td>1.35</td>
</tr>
</tbody>
</table>

The results of SLAM simulation obtained with the HI filter (with a layout similar to Figure 2) are shown in Figures 3, and 4; the actual and estimated SLAM trajectories are in Figure 3, and the time history match of the actual and estimated SLAM trajectories, the three states, obtained with the HI filter is depicted in the left-subplot of Figure 4. The right-subplot of Figure 4 shows the state-error time histories for the nominal values of Q, R, and gamma of 1.05 in the HI filter. As can be seen from Figures 3 and 4, and Table II that very accurate estimation of the LMs positions has been obtained, and hence, the results of the application of HI filter to the SLAM are found to be very encouraging.

Fig. 2 Typical way points (circles in red joined by lines in blue) and LMs (starred points in green) for the HIF based SLAM.

Fig. 3 The actual and estimated SLAM trajectories obtained with the HIF (for a layout similar to as shown in Figure 2); here green lines are waypoints. [LMs: actual given as * in blue; estimated LM positions as + in red; SLAM trajectory in x-y plane: actual as thick line, estimated as thin line -> both almost merging. For nominal Q, R, and gamma =1.05 in HIF].

www.ijaetmas.com
III. ROBOT PATH PLANNING

A mobile robot or any such mobile vehicle is required to move from an initial point, say A, to an intermediate point, and then to the final goal point, say B; this is the domain of path planning problem (PP). If there are obstacles on the path, the mobile robot/vehicle should be able to avoid these without touching these obstacles; these obstacles in the path–space/configuration space could be static or dynamically moving. The robot starts from point A and then moves ahead (the aspect of motion–trajectory planning, MP, not considered here presently) avoiding any obstacles coming in its path and then reaching its goal point B. It is most probable that there could be more than one path for a robot to follow, these paths being equally feasible paths, but only one would be an optimal path; in that case the robot should be programmed such that it follows this optimal (or at least sub-optimal) path for its travel to the goal position B. So, the immediate problem is to find a good path that the robot should follow to reach its goal and that to while ensuring that the robot’s system spends minimum energy/effort on the part of the mobile robot. An autonomous mobile system or robot has to make its own decisions, perhaps and most probably the instructions might be embedded in the on-board computer, wherefrom a suitable decision to achieve its goal or next move could originate – this move will then direct an actuator to impart motion to the robot, and area of motion planning. A major an important component of such an autonomous system is its path planner, and following the assigned optimal path gets the robot from one location or its current state to another location, and eventually to the final goal point; while simultaneously avoiding the obstacles. Path and motion planning (PMP) is also used by the transport engineers along with traffic-flow mathematical models to model the flow of traffic, and the (video) games (playing/sports games) also use some form of path planning to move computer-controlled players around the game environment, requiring a fast path planner, since multiple paths need to be planned at the given time instances.

3.1 PATH PLANNING SCENARIO

Assume that we have a robot arena with an overhead vision-camera which can be easily calibrated and the image coming from it can be used to create a robot map, say on a horizontal plane in the robot configuration/workplace. It is assumed that such a map has been given as an input to the algorithm/program [11]. It is also assumed that the coordinates of the source (starting point) and goal (ending point) of the robot are supplied. The aim is only to plan a path for the robot. In order to move a robot on the desired path, a suitable motion planning control algorithm would be require.
3.2 PATH PLANNING ALGORITHMS

The robot is considered as a point object to enable a quick implementation. We briefly discuss several PP algorithms available in the literature with some associated and pertinent results [11]. Since, there is also a great scope of application of soft computing techniques to robot path planning problem, we evaluate such two approaches based on fuzzy logic and genetic algorithm. Our idea is to evaluate these results in a composite way and suggest a performance acceptance metric, this was, perhaps, not done in many earlier studies [11].

A* Algorithm (ASA)

For the ASA, it is necessary to model the PP problem as a standard graph search algorithm. In the A* algorithm a graph is taken as an input. Then, the algorithm explores all the possible regions, and finds the shortest path from the starting-source to all states in the given regions to the goal state. It explores all the near regions, and this act is biased towards the regions closer to the goal indicated by some heuristic function. A* algorithm works in discrete spaces; the map is taken as an array of discrete pixels. The higher the resolution of the map, better are the results, however, the computation time would be excessive. The graph for search consists of vertices and edges, and each pixel of the map is taken as a vertex; and each vertex has a number of connections which act as edges, and these connections in turn are assumed collision free. The possible connections are coded as a matrix and this is given as an input to the graph; all possible moves are denoted by 1 and all impossible moves are denoted by 0. The connection matrix is an input parameter and a robot can take: i) rectilinear moves (up, down, left and right), ii) four diagonal moves along with the 4 rectilinear moves, or iii) more flexible moves adding connections between the diagonal moves; and more the number of ones in the matrix, more is the flexibility of turns for the robot, and better is the path; however, the latter would result in greater computational costs. The design specifications are: i) the weights of the edges (taken as the Euclidian distance between the connecting points, and ii) the heuristic function denoting the nearness of the point to the goal which is also taken as the Euclidian distance to the goal.

Artificial potential field (APF)

APF based navigation is a reactive planning approach, in which the immediate distances from the obstacles are considered to compute the immediate move, these actions lead to motion of the robot ultimately leading to the goal; here all the obstacles repel the robot with a magnitude inversely proportional to the distance; whereas the goal attracts the robot. In APF algorithm, the resultant potential that accounts for the attractive and repulsive components is measured and this used to move the robot. The potential field is generally indicated as arrows away from the obstacles, and the distance of the obstacles at all angles from the robot is measured. We would consider only 5 distances at specific angles measured to compute the repulsive potential: forward, left side, right side, forward left diagonal and forward right diagonal.

Rapidly exploring random tree (RERT)

The RERT algorithm grows/expands and maintains a tree where each node of the tree is a point, state in the robot’s workspace, and the area explored by the RERT algorithm is the area occupied by the tree. Initially, the RERT algorithm starts with a tree which has source as the only node, then at each iteration, the tree is expanded by selecting a random state and expanding the tree towards that state; this expansion is carried out by extending the closest node in the tree towards the selected random state by a small step. The algorithm runs till some expansion takes the tree sufficiently near enough to the goal. The size of the step taken as an algorithm parameter: the smaller values would result in slow expansion, but finer paths or paths which can take fine turns.

Bidirectional rapidly exploring random tree (BRERT)

Like the RERT, the BRERT explores the search space using trees; however, in place of one tree going from the source to the goal, now two trees are initiated and grown: i) the first tree initiates from the source and grows towards the goal, and ii) the second tree initiates from the goal and grows towards the source; and naturally, when both the trees meet, a suitable path is found.

Probabilistic roadmap (PRM)

The PRM algorithm has two phases: i) an offline roadmap/graph building stage, and ii) an online planning (query) stage. In the first phase a small graph is randomly drawn across the robot’s workspace; wherein all the vertices, and edges of the graph should be collision-free; the idea being that a robot may use the same graph for its motion planning. In the PRM algorithm, a number of random points/states in the workspace are selected as the vertices; this should not be inside any obstacle. Higher the number of vertices or k (number of states), better would be the results with of course increased computational cost. The PRM then attempts to connect all pairs of randomly selected vertices; if any two vertices can be connected by a straight line, then this straight line is added as an edge. The online planning (query stage) aims at using the roadmap developed earlier for PP of the robot. The weights of the edges are taken as the Euclidian distance between the connecting points; and the heuristic function, that denotes the nearness of the point to the goal, is taken as the Euclidian distance to the goal.
Fuzzy logic based PP algorithm (FLPPA)

FL is conceptually based on the fact that, for an example, a variable high-temperature may mean different degree of 'high' to different people, and also it might depend on the application problem; like categorising fever, or the 'hot' engine in a car. Thus, FL is based on the notion of fuzzy sets (FS) and in FL/S partial membership is permitted. FL/S based analysis and design involves: fuzzification, evaluation of rules (If...,Then..., rules), and defuzzification. A FS A is a subset of a UOD U, where A is characterized by a membership function (MF) \( \mu_A(x) \). This MF gives the grade of membership in/for FS A. The fuzzy inference system (FIS) needs: i) fuzzification of the linguistic variables, like temperature, and assigning of a membership value in a FS, ii) production rules \( \text{If... A, Then... B, etc.,} \) and iii) defuzzification to obtain crisp value from the fuzzy variable [12-14]. The rules represent the knowledge that a human expert has of the system under analysis/design; and by defining various such knowledge production rules the knowledge is captured in some sense so that it can be integrated into the FIS. In the If... A, Then... B, rule, 'If' represents the antecedent or conditional part; and 'Then' signifies the conclusion or action (consequence) part. The rules are used to encode the empirical associations between the incoming data-patterns/feature vectors and the actions that are required to be performed by the system as a consequent part. Thus, FIS can be used for robot path planning. Fuzzy based navigation in robot PP is a reactive planning approach, where the immediate position and distances from obstacles are considered to compute the immediate move [11]; in such a case immediate actions lead to motion of the robot, ultimately leading to the goal. For using FL, we first need to select a few inputs which best represent the situation that the robot is currently placed in; then the decision of motion is made purely on the basis of these inputs. For the present case certain inputs selected are: i) distance from the obstacle in front, ii) distance from the obstacle at the front left diagonal, iii) distance from the obstacle at the front right diagonal, iv) angle between the heading direction of robot and the goal, v) distance of the robot from the goal and preferred turn; the input, preferred turn indicates whether it would be beneficial to turn clockwise or anti-clockwise. Some preferred actions are: i) if the front obstacle is far away, turn is so as to more face the goal; ii) if the front obstacle is close and a new front obstacle is encountered, turn using the side of the goal is preferred; and iii) if the front obstacle is close and the same obstacle as encountered in the previous step is found, the same turn as made previously is repeated. The FL rules are written such that the robot avoids the obstacles and aligns itself towards the goal. The FS/FIS is a result of a lot of manual tuning of the rules and MFs over a wide variety of situation and scenarios. In usage of FL for PP, a FIS is used to map the inputs to the outputs and take appropriate decisions. The initial acceleration and direction of the robot are used. The distance from the current position of the robot to the obstacles is computed. Depending on these computations, the computed steer and the direction is calculated. For example, if the same obstacle is encountered in the previous step as in the current step then the turn is taken in the same direction otherwise the turn is taken in the opposite direction. The conditions to be checked, whether the point in which the robot is turning is feasible or not: i) the point \((x,y)\) should lie within the map, and ii) the point \((x,y)\) is checked; if the value is 1, then it corresponds to a point on the obstacle. A Mamdani type FL model is used here; and the centroid defuzzification method is used [14]. In FLPPA some additional parameters like speed of the robot, direction scaling of the robot, and threshold distance to the obstacle can be set along apart from some of the common parameters in other algorithms. The threshold distance is set to 30, the direction scaling is set to 60 degrees. Various values like distance of obstacle diagonally and distance of obstacle from the front, back, and the sides are computed in this algorithm and given to the FIS to calculate the computed steer.

Genetic algorithm based PP algorithm (GAPPA)

The GAs are part of the evolutionary computation (EC) that simulates evolution on a digital computer [14]. From such a simulation emerge certain optimization algorithms which are based on certain simple rules. The goal is to obtain a feasible solution of the optimization problem. The EC mimics the biological evolution and is a search procedure that probabilistically applies search operators in the search space to obtain a robust, and perhaps an optimal solution. Based on the biological systems, GAs adopt the rules of natural selection and genetics to obtain robust solutions. In fact the solutions are supposed to be globally optimal (but not always) and robust. A simple procedure based on the premise of survival of the fittest, a population or samples of feasible solutions is combined in a manner similar to the combination of chromosomes in a natural/biological genetic system. The fitter population members/samples pass on their structures forward as genes in a far greater measure than their less fit samples do. As the generations evolve, the net effect is evolution of the population towards an optimum. We next describe briefly the constituents of a typical GA [14].

a) Chromosomes: These represent encoding of information in a string of fixed (finite) length and each one consists of a string of bits (binary digit: 0 or 1); it could be a symbol from a set of more than two elements;
generally for function optimization the chromosomes are constructed from binary strings. These chromosomes consist of genes, and each gene represents a unit of information and it can take different values. Thus, the strings composed of features or detectors, assume values like 0 or 1 that are located at different positions in the string. For real life cases, the real numbers are used for coding.

b) Population and fitness: GA operates on population of possible samples/candidate solutions with chromosomes. The population members are known as individuals/samples. Each individual/sample is assigned a fitness-value based on the objective function. Better individuals (samples/solutions) have higher fitness values and weaker ones have lower fitness-values; the latter are discarded in the next cycle.

c) Initialization and reproduction: A population of possible initial solutions is created by randomly selecting information from the search space and encoding it. In reproduction process the individual strings are copied as per their fitness values. The strings with a greater fitness-value have higher probability of contributing one or more off-springs to the next generation.

d) Crossover: In a crossover process, a site/location is selected randomly along the length of the chromosomes (i.e. onto its encoded length of bits), and each chromosome is split into two pieces at this crossover site. The new strings are formed by joining the top piece of one chromosome with the tailpiece of the other in the crossover process. Often, the crossover is performed within a chromosome sting only.

e) Mutation: In this small operation a bit (only one bit) in a string is changed at a random location (0 is flipped to 1 or 1 is flipped to 0). The idea is to break the monotony and add a bit of novelty.

f) Generation: Each iteration in the GA based optimization process is called a generation, and in each generation pairs are chosen for crossover operation, the fitness is determined, and mutation is carried out during the crossover process. Then a new population evolves which is carried forward.

g) Survival of the fittest: The individuals/samples/candidates may be fitter or weaker than some other population members. So, they must be ranked as per their fitness value. In each generation, the weaker ones are allowed to wither out and the ones with good fitness values take part further in the genetic operation. The final result is the evolution of the population towards the global optimum.

h) Cost function, decision variables and search pace: In many optimization problems, the goal is to find optimal parameters to increase the production and/or to reduce the expenditure/loss. That is done to get maximum profit by reorganizing the system/its parameters that affect the cost function. Such parameters of the system that decide the cost are called the decision variables, and this is achieved by using some norm in the Euclidean vector space.

To model the problem for GA as an optimization problem, we need an objective function and specification of variables of that objective function (along with their bounds). Let a path be characterized by a fixed number of points in the robotic map. In order to make some path from this set of points, we start from the source and connect it to the first point by a straight line. The first point is connected to the second point by a straight line, and so on [11]. At the end the last point is connected to the goal. The objective function is the length of this path. A heavy penalty is added if any part of the path lies inside the obstacle, while the penalty is proportional to the length of the path inside the obstacle. The locations of each of these fixed number of points (both X and Y axis positions) are the optimization variables. The variable bounds are such that the point lies inside the map (lower bound 1 and upper bound as the length/width of the map for the X/Y axis). Each point in the path marks a point of turn; the total number of points is an algorithm parameter and should be equal to the maximum number of turns a robot is expected to make in the robot map; setting this number too high would result in very large computational requirements. Initial inputs for the GA PP algorithm are the length of the gene and the population of the genes; also, the starting point, the end point and the obstacles are specified. In the GA, the value of the fitness function for each chromosome is computed; normally, the chromosome which yields maximum fitness function value, is said to be the winner; this is more likely to appear for the next generation rather than the chromosome that yields minimal fitness function value. The winner chromosomes are then used in the next iteration for the cross over to produce more genes. A crossover and the mutation rate are initially set and these rates dictate the cross over and the mutation between the genes. Fitness function of genetic algorithm for path planning is generally the path (length) cost; which is computed and the chromosome that yields the minimum path cost is generally selected for cross over in the next iteration. The algorithm works in such a way that a line is drawn between the starting point and one of the intermediate points. The line is then observed to see if it passes through any of the obstacles. This is done recursively until the path is planned till it reaches the goal. After the basic path is formed a spline function can be used to get the smoothened path from the starting
point to the end point. In GA one of the main inputs before proceeding to run the algorithm is the number of intermediate points. These are set such that it lies in the map and does not coincide with any obstacle. The number of intermediate points has to be chosen appropriately. Generally, for a map with less but large obstacles the number of these points is less. But in case where there are lots of small obstacles, the number of intermediate points is needed to be very high. If the value is not set appropriately, the path will not be planned and may also result in not yielding any path from the source to the goal in the map. Also the number of generations of obstacles over which the path will be planned is set to 10 in this case; and the number of chromosomes i.e. the population is set to 50; and in general these numbers are user specified.

3.3 REVIEW OF THE PP ALGORITHMS’ RESULTS AND A COMPOSITE ASSESSMENT

General procedure to generate the result using the PP algorithms discussed in the preceding sections is [11]:

i) First make a ‘bitmap’ file and draw any map on it, over which you would like to execute the chosen PP algorithm; for this ‘Paint’ or any other simple drawing tool may be used; and then save the file as a BMP; place the file in the project folder.

ii) Alter the name of the map file in the chosen code to point out to the map that has been created

iii) Supply the source and the goal positions; one can use paint or any other drawing utility which displays the pixel positions of the points, to locate the source and goal on the drawn map.

iv) Alter the X and Y resolutions as required.

v) Choose the required parameters (e.g. connection matrix for A* algorithm; k for PRM algorithm, step size for RERT/BRERT; number of points, population size, number of iterations for Gappa; initial heading for FLPPA, etc.) for the given PP algorithm. vi) Execute the script astart.m of the chosen PP algorithm. Note down the numerical results

The path planning algorithms were evaluated [11] on the basis of time taken for the computation of the feasible path and the length of the path from the start point to the end point. The algorithms were also compared with the method of allowing a robot to move from the start point to the end point and making a turn in the presence of obstacle. Two cases were considered: i) containing fewer obstacles, and ii) containing more obstacles in the configuration space. Various results [11] are shown in Table II, in a composite way to further analyse in subsequent paragraphs; however, the following immediate observations are made:

i) For A* PP algorithm: source and goal points are: [10 10]; and [490 490]; the resolution of the original map: 500 x 500; and the lower resolution as 100 x 100. The increase in the connection matrix would result in better paths. For more connections the execution time would also increases. The path cost is high even for a few connection points, because of the peculiar disposition of the obstacles.

ii) For potential field PP algorithm: source and goal points: [50 50]; and [450 450]; the resolution of the original map: 500 x 500. The costs, i.e. the path length in most cases are comparable; the largest execution time is for the three rectangular obstacles, due to perhaps the disposition of two nearing such obstacles.

For RERT PP algorithm: source and goal points are: [10 10]; and [490 490]; the resolution of the original map: 500 x 500. Even with three obstacles, the path cost is relatively high, since the disposition of these obstacles is somewhat peculiar; otherwise there is a trend between the number of obstacles and the execution time.

For BRERT PP algorithm: the parameters are the same as the ones for the RERT algorithm. It can be seen that for the same trees the path costs are somewhat larger in some cases. The execution times are relatively lower for RERT and BRERT algorithms compared to the A* PP and Pfpp algorithms.

For PRM PP algorithm: the parameters are the same as the ones for the RERT algorithm. The execution times are relatively greater than those for the preceding algorithms.

For FL PP algorithm: source and goal points are: [20 20]; and [480 480]; the resolution of the original map: 500 x 500. The path length/cost and execution times are just reasonable, and do not vary much with the number of obstacles.

For GA PP algorithm: source and goal points are: [10 10]; and [490 490]; the resolution of the original map: 500 x 500; the population size is 50, and the number of generations is 10. As expected, the execution times are very large compared to the preceding PP algorithms.
Next, in order to further analyze the results presented in Table II, we arrange the two metrics, in Table III, the path cost and the execution time (ET in msec.) according to the configurations 1 to 5 that are in somewhat increasing level (dimensions) of the connection matrix and/or more number of obstacles; in a composite way. We see that except for the A* algorithm, there is no trend of the ET wrt the configurations. Interestingly the path cost for the PFA remains almost constant. Execution times for the BRERT remain the lowest for almost all the configurations compared to the other PP algorithms. Interestingly, the ETs and path costs for the FL based PP algorithm remain nearly similar across the configurations. Somewhat similar, but not exactly, happens for the GA based PP algorithm.

So, a heuristic and composite performance acceptance metric (PAM) for selecting a relatively better PP algorithm is suggested as

$$PAM = \frac{w_1(j) \times PLC(j) + w_2(j) \times ET(j)}{w_1(j) + w_2(j)}$$

(8)

In (8), \(w(.)\) are the weights that can give relative weightage (if required), and also the \(w\)'s take

**TABLE II**

THE RESULTS OF SEVERAL ROBOT PP ALGORITHMS

<table>
<thead>
<tr>
<th>A*algorithm parameters</th>
<th>Expansion</th>
<th>Path</th>
<th>Time taken (ms)</th>
<th>Cost of path finding</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connection Matrix:</strong></td>
<td>[1 1 1; 1 2 1; 1 1 1]</td>
<td><img src="image1.png" alt="Image" /></td>
<td>1580</td>
<td>775</td>
</tr>
<tr>
<td>[1 1 1; 1 2 1; 1 1 1]</td>
<td><img src="image2.png" alt="Image" /></td>
<td>2170</td>
<td>1045</td>
<td></td>
</tr>
<tr>
<td>[1 1 1 1; 1 1 1 1; 1 1 2 1; 1 1 1 1; 1 1 1 1]</td>
<td><img src="image3.png" alt="Image" /></td>
<td>5010</td>
<td>890</td>
<td></td>
</tr>
<tr>
<td>[1 1 1 1 1; 1 1 1 1 1; 1 1 2 1 1; 1 1 1 1 1; 1 1 1 1 1]</td>
<td><img src="image4.png" alt="Image" /></td>
<td>2830</td>
<td>702</td>
<td></td>
</tr>
<tr>
<td>[0 1 0; 1 2 1; 0 1 0]</td>
<td><img src="image5.png" alt="Image" /></td>
<td>1780</td>
<td>1060</td>
<td></td>
</tr>
<tr>
<td><strong>Potential field algorithm</strong></td>
<td></td>
<td></td>
<td>2680</td>
<td>785</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2840</td>
<td>839</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>2150</td>
<td>775</td>
<td>1850</td>
<td>775</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2940</td>
<td>779</td>
<td></td>
</tr>
<tr>
<td><strong>RERT Algorithm Trees and paths→</strong> (step size 20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>892</td>
<td>846</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>501</td>
<td>1301</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>928</td>
<td>1055</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>315</td>
<td>814</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>474</td>
<td>1057</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BRERT Algorithm-Trees and paths→</strong> (step size 20)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>93</td>
<td>933</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>344</td>
<td>1225</td>
<td></td>
</tr>
<tr>
<td>Algorithm</td>
<td>k=50</td>
<td>k=100</td>
<td>k=100</td>
<td>k=50</td>
</tr>
<tr>
<td>--------------------</td>
<td>------</td>
<td>-------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>PRM Algorithm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(k=50)</td>
<td>954</td>
<td>93</td>
<td>173</td>
<td>3190</td>
</tr>
<tr>
<td>Road Maps and Paths</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GA based Algorithm (Number of points=3)</td>
<td>ET (sec.)</td>
<td>Source</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>-----------</td>
<td>--------</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1013</td>
<td>899</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>997</td>
<td>901</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>900</td>
<td>898</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>947</td>
<td>900</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>108 (sec.)</td>
<td>929</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>111 (sec.)</td>
<td>1312</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>121 (sec.)</td>
<td>928</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>142 (sec.)</td>
<td>982</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>114 (sec.)</td>
<td>931</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The ETs for RERT, BRERT and PRM were separately recorded, and hence, do not correspond to the plots shown [11]; Source: adapted/modified from [11] for the purpose of composite evaluation with a novel metric.
care of the differing units of the path (length) cost, PLC and the ET; we retain ET in msec; j is the index representing a configuration number. Applying the metric, (8), for the entries in Table III, and presently considering equal weightages, we obtain the entries as in the last column of Table III, on the average for all the configurations. Thus, based on PAM, looking for the lowest value, we can readily observe that amongst the classical algorithms, the BRERT is the best and the PRM is the worst; and the A* and PF are of comparable acceptance. The RERT is also reasonably comparable to BRERT, though not exactly. Amongst the soft computing based PP algorithms, the best is the FL based PP algorithm and GA fairs badly. Since, the PLCS are reasonably comparable across the rows for each configuration, in the present case the PAM as defined in (8) becomes the guiding factor. If needed, the metric (8) can be separately evaluated for PLC and ET.

### IV. CONCLUSIONS

The problem of simultaneous localization and mapping for a mobile vehicle has been studied and we have presented a relatively novel application of H-infinity posterior filter for the SLAM problem in the area of navigation and guidance for such a mobile vehicle. From the numerical simulation results for the trajectory and landmarks estimation, we can infer that the performance of the H-\( \infty \) filter for SLAM has been very satisfactory and encouraging. We have also comprehensively reviewed several algorithms for robot path planning (and their associated performance results) that can be considered as a special case of the general SLAM problem, i.e. robot localization is a degenerate or the point case of the entire PP scenario. Based on a very simple, but sort of a novel fusion metric computed using the execution times and the path costs, averaged on the variety of connection matrices and/or number of obstacles and their disposition, in a composite way, we infer that, amongst the classical algorithms evaluated, the bidirectional rapidly exploring random tree is the best (BRERT algorithm), and compared to the GA PP algorithm, the FL PP algorithm generally does better. The results of the present study could pave the way for more applications of the H infinity based robust filtering algorithms to multi-robot SLAM. Also, the performance evaluation metric, presented and validated here, can be further evaluated/utilized for comparing the robot path planning algorithms in multi-robot coordination scenarios.

### ACKNOWLEDGEMENTS:

The first author is very grateful to his guide, Dr. Sheshachalam, D., Professor, & Head, Dept. of E&CE, BMS College of Engineering, Bangalore, for technical discussions, fruitful guidance and encouragement.

### REFERENCES


