Unit IV Navigation, Path Planning and Control Architecture

4.1 Localization and Navigation

Robot localization is the process of determining where a mobile **robot** is located with respect to its environment. **Localization** is one of the most fundamental competencies required by an autonomous **robot** as the knowledge of the **robot's** own location is an essential precursor to making decisions about future actions.

Encoder: an encoder is a sensing device that provides feedback. Encoders convert motion to an electrical signal that can be read by some type of control device in a motion control system, such as a counter or PLC. The encoder sends a feedback signal that can be used to determine position, count, speed, or direction. A control device can use this information to send a command for a particular function.

Position prediction:

The robot then collects actual sensor data and extracts appropriate features (e.g. lines, doors, or even the value of a specific sensor) in the observation step. At the same time, based on its predicted position in the map, the robot generates a measurement prediction which identifies the features that therobot expects to find and the positions of those features.

Observation:

The observation is the result of a feature extraction process executed on the raw sensor data.

Matching:

In matching the robot identifies the best pairings between the features actually extracted during observation and the expected features due to measurement prediction

The parameters of the features are usually specified in the sensor frame and

therefore in a local reference frame of the robot. However, for matching we need to represent the observations and measurement predictions in the same frame.

Estimation:

Next we compute the best estimate of the robot's position based on the position prediction and all the observations.There are two methods for prediction of position;

Odometry *:* Use wheel sensors to update position

Dead Reckoning**:** Use wheel sensors and heading sensor to update position Odometry Error Sources:

- Limited resolution during integration
- Unequal wheel diameter
- Variation in the contact point of the wheel
- Unequal floor contact and variable friction can lead to slipping
- Deterministic errors can be eliminated through proper calibration

Non-deterministic errors have to be described by error models and will always lead to uncertain position estimate.

From a geometric point of view one can classify the errors into three types:

• *Range error:*integrated path length (distance) of the robots movement -> sum of the wheel movements

• *Turn error*: similar to range error, but for turns -> difference of the wheel motions

- *Drift error:* difference in the error of the wheels leads to an error in the robot's angular orientation

In navigation, **dead reckoning** is the process of calculating one's current position by using a previously determined position, or fix, and advancing that position based upon known or estimated speeds over elapsed time and course.

- Heading sensors can be proprioceptive (gyroscope, inclinometer) or exteroceptive(compass).
- Used to determine the robots orientation and inclination.
- Allow, together with an appropriate velocity information, to integrate the movementto an position estimate.

This procedure is called dead reckoning (ship navigation)

4.2 Probabilistic based Localization

There are two classes of probabilistic localization.

The first class, Markov localization, uses an explicitly specified probability distribution acrossall possible robots positions.

The second method, Kalman filter localization, uses a Gaussian probability density representation of robot position and scan matching for localization. Unlike Markov localization, Kalman filter localization does not independently consider each possible pose inthe robot's configuration space.

Markov localization allows for localization starting from any unknown position and can thus recover from ambiguous situations because the robot can track multiple, completely disparate possible positions. However, to update the probability of all positions within the whole state space at any time requires a discrete representation of the space (grid). The required memory and computational power can thus limit precision and map size.

Kalman filter localization tracks the robot from an initially known position

and is inherently both precise and efficient. In particular, Kalman filter localization can be used in continuous world representations. However, if the uncertainty of the robot becomes too large (e.g. due to a robot collision with an object) and thus not truly unimodal, the Kalman filter can fail to capture the multitude of possible robot positions and can become irrevocably lost.

The key difference between the Kalman filter approach and our earlier Markov localization approach lies in the perception update process. In Markov localization, the entire perception,i.e. the robot's set of instantaneous sensor measurements, is used to update each possible robot position in the belief state individually using Bayes formula. In some cases, the perception is abstract, having been produced by a feature extraction mechanism as in Dervish. In other cases, as with Rhino, the perception consists of raw sensor readings. By contrast, perception update using a Kalman filter is a multi-step process. The robot's total sensory input is treated, not as a monolithic whole, but as a set of extracted features that each relate to objects in the environment. Given a set of possible features, the Kalman filter is used to fuse the distance estimate from each feature to a matching object in the map. Instead of carrying out this matching process for many possible robot locations individually as in the Markov approach, the Kalman filter accomplishes the same probabilistic update by treating the whole, unimodal and Gaussian belief state at once.

4.3 NAVIGATION:

The specific aspect of cognition directly linked to robust mobility is navigation competence. Given partial knowledge about its environment and a goal position or series of positions, navigation encompasses the ability of the robot to act based on its knowledge and sensor values so as to reach its goal positions as efficiently and as reliably as possible

Localization versus Navigation:

In creating a navigation system, it is clear that the mobile robot will need sensors and a motion control system. Sensors are absolutely required to avoid hitting moving obstacles such as humans, and some motion control system is required so that the robot can deliberately move.

Two methods of Navigation:

***Behavior based Community*:**

An alternative, espoused by the behavior-based community, suggests that, since sensors and effectors are noisy and information-limited, one should avoid creating a geometric map for localization.

This approach avoids explicit reasoning about localization and position, and thus generally avoids explicit path planning as well.

This technique is based on a belief that there exists a procedural solution to the particular navigation problem at hand. For example, in Fig. 4.1, the behavioralistic approach to navigating from Room A to Room B might be to design a left-wall-following behavior and a detector for Room B that is triggered by some unique queue in Room B, such as the color of the carpet. Then, the robot can reach Room B by engaging the left wall follower with the Room B detector as the termination condition for the program

Fig.4.1 A Sample environment

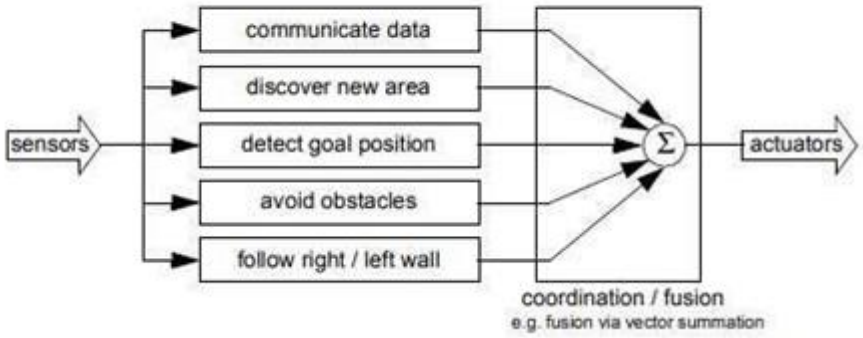


Fig. 4.2 An Architecture for Behaviour based (Model based) Navigation

The architecture of this solution to a specific navigation problem is shown in figure 4.2. The key advantage of this method is that, when possible, it may be implemented very quickly for a single environment with a small number of goal positions.

It suffers from some disadvantages, however. First, the method does not directly scale to other environments or to larger environments. Often, the navigation code is location-specific, and the same degree of coding and

debugging is required to move the robot to a new environment.

In contrast to the behavior-based approach, ***the map-based approach*** includes both localization and cognition modules (see Fig. 4.3). In map-based navigation, the robot explicitly attempts to localize by collecting sensor data, then updating some belief about its position with respect to a map of the environment. The key advantages of the map-based approach for navigation are as follows:

- The explicit, map-based concept of position makes the system's belief about position transparently available to the human operators.

• The existence of the map itself represents a medium for communication between human and robot: the human can simply give the robot a new map if the robot goes to a new environment.

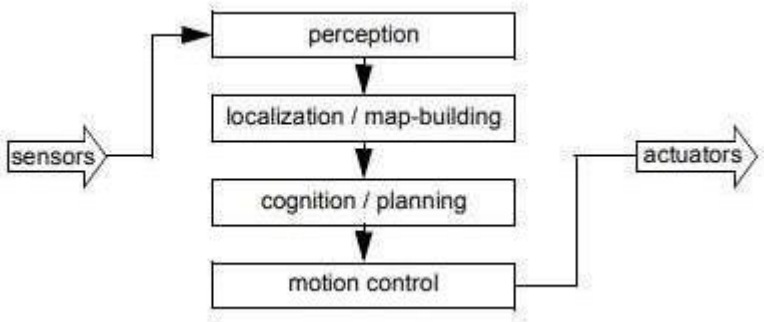


Fig. 4.3 An Architecture for Map based (Model based) Navigation

The map, if created by the robot, can be used by humans as well, achieving two uses. The map-based approach will require more up-front development effort to create a navigating mobile robot.

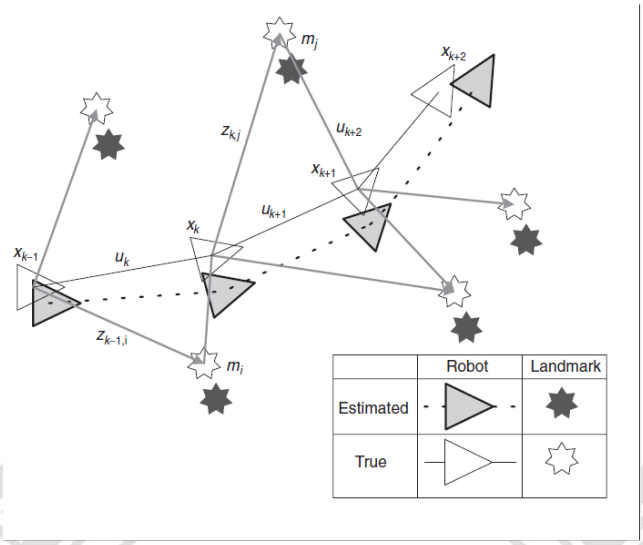
4.4 Simultaneous Localizationand Mapping (SLAM)

The simultaneous localization and mapping (SLAM) problem asks if it is possible for a mobile robot to be placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map. A solution to the SLAM problem has been seen as a "holy grail" for the mobile robotics community as it would provide the means to make a robot truly autonomous.

The "solution" of the SLAM problem has been one of the notable successes of the robotics community over the past decade. SLAM has been formulated and solved as a theoretical problem in a number of different forms. SLAM has also been implemented in a number of different domains from indoor robots to outdoor, underwater, and airborne systems. At a theoretical and conceptual level, SLAM can now be considered a solved problem. However, substantial issues remain in practically realizing more general SLAM solutions and notably in building and using perceptually rich maps as part of a SLAM algorithm.

Formulation and Structureof the SLAM Problem

SLAM is a process by which a mobile robot can build a map of an environment and at the same time use this map to deduce its location. In SLAM, both the trajectory of the platform and the location of all landmarks are estimated online without the need for any a priori knowledge of location.



SLAM problem. A simultaneous estimate of both robot and landmark locations is required. The true locations are never known or measured directly. Observations are made between true robot and landmark locations.

Preliminaries

Consider a mobile robot moving through an environment taking relative observations of a number of unknown landmarks using a sensor located on the robot as shown in above Figure. At a time instant *k*, the following quantities are defined:

◆**x***k*: the state vector describing the location and orientation of the vehicle

◆**u***k*: the control vector, applied at time $k-1$ to drive the vehicle to a state **x***k* at time *k*

m*i*: a vector describing the location of the *i*th landmark whose true location is assumed time invariant

◆**z***ik* : an observation taken from the vehicle of the location of the *i*th landmark at time *k*. When there are multiple landmark observations at any one time or when the specific landmark is not relevant to the discussion, the observation will be written simply as **z***k*.

In addition, the following sets are also defined:

◆**X**0:*k* = {**x**0, **x**1, · · · , **x***k*} = {**X**0:*k*−1, **x***k*}: the history of vehicle locations

◆**U**0:*k* = {**u**1, **u**2, · · · , **u***k*} = {**U**0:*k*−1, **u***k*}: the history of control inputs

◆**m** = {**m**1,**m**2, · · · ,**m***n*} the set of all landmarks

◆ $\mathbf{Z}0{:}k = \{\mathbf{z}1, \mathbf{z}2, \cdots, \mathbf{z}k\} = \{\mathbf{Z}0{:}k-1, \mathbf{z}k\}$: the set of all landmark observations.

Probabilistic SLAM

In probabilistic form, the simultaneous localization and map building (SLAM) problem requires that the probability distribution

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) \quad (1)$$

be computed for all times k. This probability distribution describes the *joint* posterior density of the landmark locations and vehicle state (at time k) given the recorded observations and control inputs up to and including time k together with the initial state of the vehicle. In general, a recursive solution to the SLAM problem is desirable. Starting with an estimate for the distribution $P(\mathbf{x}_{k-1}, \mathbf{m}\ \mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1})$ at time k-1, the joint posterior, following a control $\mathbf{u}_k$ and observation $\mathbf{z}_k$, is computed using Bayes theorem. This computation requires that a state transition model and an observation model are defined describing the effect of the control input and observation respectively.

The *observation model* describes the probability of making an observation $\mathbf{z}_k$ when the vehicle location and landmark locations are known and is generally described in the form

$$P(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m}). \quad (2)$$

It is reasonable to assume that once the vehicle location and map are defined, observations are conditionally independent given the map and the current vehicle state.

The *motion model* for the vehicle can be described in terms of a probability distribution on state transitions in the form

$P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k)$ (3)

That is, the state transition is assumed to be a Markov process in which the next state $\mathbf{x}_k$ depends only on the immediately preceding state $\mathbf{x}_{k-1}$ and the applied control $\mathbf{u}_k$ and is independent of both the observations and the map.

The SLAM algorithm is now implemented in a standard two-step recursive (sequential) prediction (time-update) correction (measurement-update) form:

Time-update

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0) = \int P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k) \times P(\mathbf{x}_{k-1}, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}, \mathbf{x}_0)d\mathbf{x}_{k-1} \quad (4)$$

Measurement Update

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) = \frac{P(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m})\, P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0)}{P(\mathbf{z}_k|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k})} \quad (5)$$

Equations (4) and (5) provide a recursive procedure for calculating the joint posterior $P(\mathbf{x}k,\mathbf{m}|\mathbf{Z}0{:}k,\mathbf{U}0{:}k, \mathbf{x}0)$ for the robot state $\mathbf{x}k$ and map $\mathbf{m}$ at a time k based on all observations $\mathbf{Z}0{:}k$ and all control inputs $\mathbf{U}0{:}k$ up to and including time k. The recursion is a function of a vehicle model $P(\mathbf{x}k|\mathbf{x}k-1, \mathbf{u}k)$ and an observation model $P(\mathbf{z}k|\mathbf{x}k,\mathbf{m})$.

It is worth noting that the map building problem may be formulated as computing the conditional density $P(\mathbf{m}|\mathbf{X}0{:}k,\mathbf{Z}0{:}k,\mathbf{U}0{:}k)$. This assumes that the location of the vehicle $\mathbf{x}k$ is known (or at least deterministic) at all times,

subject to knowledge of initial location. A map **m** is then constructed by fusing observations from different locations.

Conversely, the localization problem may be formulated as computing the probability distribution $P(\mathbf{x}k|\mathbf{Z}0: k,U0:k,m)$. This assumes that the landmark locations are known with certainty, and the objective is to compute an estimate of vehicle location with respect to these landmarks.

Solutions to the SLAM Problem

Solutions to the probabilistic SLAM problem involve finding an appropriate representation for both the observation model (2) and motion model (3) that allows efficient and consistent computation of the prior and posterior distributions in (4) and (5). By far, the most common representation is in the form of a state-space model with additive Gaussian noise, leading to the use of the extended Kalman filter (EKF) to solve the SLAM problem. One important alternative representation is to describe the vehicle motion model in (3) as a set of samples of a more general non- Gaussian probability distribution. This leads to the use ofthe Rao-Black wellized particle filter, or Fast SLAM algorithm, to solve the SLAM problem. While EKF-SLAM and Fast SLAM are the two most important solution methods, newer alternatives, which offer much potential, have been proposed, including the use of the information-state form.

4.5 Optimization Algorithms for Robot Navigation

There are many different types of navigation algorithms that a robot may use to traverse space. Often these algorithms have similar processes but differ only slightly in different situations. For example, when a robot hits a dead end, where does it go next? Two different algorithms could have gotten the robot there using the exact same path, but differ by how they choose the next step.

Another way two algorithms could differ is in how they choose the path back to the original start state after it has found the goal.

There to be three specific situations where there is a major difference between navigation algorithms when choosing a path. Because of these situations, we chose to split the algorithms into three different steps.

The first step deals with the generic situation, how to begin and continue the traversal of unknown space. This step chooses which one of the neighboring cells to visit.

The second situation is used when the first situation cannot pick an unvisited cell to visit. This occurs when either, all the adjacent unvisited cells are blocked by walls, or if all the adjacent cells have already been visited. At this point, a decision needs to be made by the robot. It needs to choose a path to continue its search of the unknown environment.

The third step is when the robot has found the goal and needs to proceed back to the initial starting point. There are several different ways that the robot may choose to return to the origin.

There are several algorithms implemented for each of the steps. For the first step there are two different algorithms. One is called **"Follow the Right Wall."** This consists of the robot following a wall similarly to a person walking along a path with their hand always touching the wall on that side. A wall can be substituted with a cell that has already been visited.

The second algorithm implemented for the first step is called "Right Left Sweep." This algorithm is similar to "Follow the Right Wall" except when the robot enters a cell that is adjacent to a boundary wall, the direction switches. For example, if the robot is currently following the right wall and enters a cell with a boundary wall the robot would start following the left

wall. This would continue until it enters another cell that is bordered by a boundary wall.

The implemented algorithms for the second step are as follows. The first implementation is called “Find Closest Unvisited Cell.” This algorithm will consider the cells the robot has not yet been in, and pick the one that is closest to the robot. The second algorithm is “Find Closest Unvisited Cell to Start.” This is similar to the first algorithm except that the base reference point is the start instead ofthe robot’s current position.

The third algorithmic step, returning to the start after finding the goal, also has multiple implementations. The first implementation is “Assume No Walls.” This is an optimistic path finding algorithm that will plot a path back to the start assuming that all unknown potential walls will not have walls. If a wall is found along the return path, a new path needs to be calculated. The second is “Assume Walls.” This is a pessimistic algorithm that will plot a path back to the start using only the information gathered. All the unknowns are assumed to be a wall when traveling back to the start with this algorithm. A new path will never need to be calculated because the return path is based upon a path the robot knows exists. The third is “Assume Walls Unless Cell is Visited.” This is in between the previous two algorithms. It will only assume an unknown is not a wall if the cell it is attempting to enter has already been visited. Otherwise it will assume that all unknowns are walls.

Twelve different algorithms can be created with the different combinations of the above steps. We number the algorithms for later reference.

Algorithm Number	Left Right Sweep	Follow Right Wall	Closest to Robot	Closest to Start	Assume No Walls	Assume Walls	No Walls if Visited
1	X		X		X		
2		X	X		X		
3	X			X	X		
4		X		X	X		
5	X		X			X	
6		X	X			X	
7	X			X		X	
8		X		X		X	
9	X		X				X
10		X	X				X
11	X			X			X
12		X		X			X

The results from these tests were split into three different categories. The robot logged the number of bumps, turns and moves it took during the duration of each test. With these different statistics we can apply weights to each type of movement and anticipate the amount of time it would take different robots to reach the goal in the same situation. The weights are multipliers applied to the different movement types. We used three different sets of weights against our results, each adding up to a total of 15. The first weight applied was five on each of the three movement types. This was to model the robot that takes the same amount of time to do all three movements. The second is three on one and six on the other two. This set demonstrates the robots that are quick at one of the movements and slower at the other two. The third is nine on one and three on the other two. This weight set describes the robots that are really slow at one of the movements and fast at the other two movements.