Automated Rover Unguided Navigation System Analysis

By Shaonak Dayal

Introduction

The entire system can be divided into the following subsystems

- Drift and Bias Correction
    - The drift in the sensor values is compensated using extended Kalman filter
- Rover Localisation
    - Using inertial motion sensors and GPS, estimate rover's position on a global map
- Camera stereo input
    - Getting stereo input images from two forward cameras to be used in a V-disparity map
    - A Kinect camera senor would be an good choice for such an application
- Obstacle detection
    - Using a V-disparity map approach, detect obstacles in the rover's path
- Motion planning
    - For navigating through the obstacles, an A* motion planning algorithm is used
- Contour detection
    - For detecting the circular contour marker
- Motion control
    - Controlling the driving motors to move the rover in the desired direction

Drift and Bias Correction

The accelerometer, gyroscope and magnetometer sensors are part of the inertial measurement unit (IMU) which give the pitch, roll and yaw sensor values.

The drift in the sensor values of the IMU and the bias in the GPS and the IMU are corrected using an extended Kalman filter.

In the extended Kalman filter, the state transition and observation models don't need to be linear functions of the state but may instead be differentiable functions.

$$xk = f(x(k-1), uk) + wk$$

$$zk = h(xk) + vk$$

Here wk and vk are the process and observation noises which are both assumed to be zero mean Gaussian noises with covariance Qk and Rk respectively. uk is the control vector.

The function f is used to compute the predicted state from the previous estimate and similarly the function h is used to compute the predicted measurement from the predicted state. However, f and h can't be applied to the covariance directly. Instead a matrix of partial derivatives (the Jacobian) is computed.

At each time step, the Jacobian is evaluated with current predicted states. These matrices can be used in the Kalman filter equations. This process essentially linearizes the non-linear function around the current estimate.

Model forecast

$$\begin{aligned}\mathbf{x}_k^f &\approx \mathbf{f}(\mathbf{x}_{k-1}^a)\\ \mathbf{P}_k^f &= \mathbf{J_f}(\mathbf{x}_{k-1}^a)\mathbf{P}_{k-1}\mathbf{J_f}^T(\mathbf{x}_{k-1}^a) + \mathbf{Q}_{k-1}\end{aligned}$$

Corrector

$$\begin{aligned}\mathbf{x}_k^a &\approx \mathbf{x}_k^f + \mathbf{K}_k(\mathbf{z}_k - \mathbf{h}(\mathbf{x}_k^f))\\ \mathbf{K}_k &= \mathbf{P}_k^f\mathbf{J_h}^T(\mathbf{x}_k^f)\left(\mathbf{J_h}(\mathbf{x}_k^f)\mathbf{P}_k^f\mathbf{J_h}^T(\mathbf{x}_k^f) + \mathbf{R}_k\right)^{-1}\\ \mathbf{P}_k &= \left(\mathbf{I} - \mathbf{K}_k\mathbf{J_h}(\mathbf{x}_k^f)\right)\mathbf{P}_k^f\end{aligned}$$

Obstacle detection and Rover Localisation

With stereo images from the cameras, objects in the cameras' field of view will appear at slightly different locations in the two images due to the cameras' different positions/perspectives on the scene.
The data from the IMU is used to create a semi global bundle adjustment for rover pose and 3D point cloud. The GPS data along with the local states, represent the rover's position in the terrain as well as on the global map
The obstacles are obtained by subtracting the ground plane from the original disparity map
Depth information is computed from a pair of stereo images by first computing the distance in pixels between the location of a feature in one image and its location in the other image. This gives us a "disparity map". It looks a lot like a depth map because pixels with larger disparities are closer to the camera, and pixels with smaller disparities are farther from the camera. Each row of the V-disparity image is a matrix of various disparity values that appeared on that row in the disparity map. The disparities of the points on the ground plane appear as a strong line in the V-disparity map.

- A disparity map IΔ has been computed from the stereo image pair. Let H be the function of the image variable IΔ such that H(IΔ) = IvΔ. Take IvΔ as the "v-disparity" image ,H accumulates the points with the same disparity that occur on a given image line i. For the image line i, the abscissa uM of a point M in IvΔ corresponds to the disparity ΔM and its grey level iM to the number of points with the same disparity ΔM on the line

  i : iM = ∑P∈IΔ δvP , iδΔP, ΔM

  where δi,j denotes the Kronecker delta. Once IΔ has been determined, IvΔ is built by accumulating the pixels of same disparity in IΔ along the v axis

- For a given image line, the grey level of a point in IvΔ expresses the number of coherent points with disparity ΔM in the same image line of the pair of given stereo images. The construction of IvΔ in facts amounts to a global analysis of an image line in IGr : we determine how this image line matches the same image line in IGr , for different horizontal offsets, that is to say for different disparities. In this manner, the horizontal order constraint is satisfied and the matching is semi-global (global for an image line).
- Let P be a point (X, Y, Z, 1)` in Ra. The ordinate of the projection of this point on the left or right image is
  vl = vr = v , v = [v0 sin θ + α cos θ](Y+ h)+[v0 cos θ − α sin θ]Z (Y + h) sin θ + Z cos θ
  Moreover, the disparity Δ of the point P is:
  Δ = ul − ur = αb (Y + h) sin θ + Z cos θ
  From the above equations, the plane with the equation Z = aY +d in Ra is projected along the straight line of the below equation in the "v-disparity" image:
  ΔM = b ah − d (v−v0)(a cos θ+sin θ)+ b ah − d α(a sin θ−cos θ)
  Thus, a surface which is formed by a succession of parts of planes is therefore projected as a piecewise linear curve.

Motion Planning

Once the obstacles have been determined, an A* pathfinding algorithm is used to obtain an optimum path for the rover to take.

A* (A star) is a search algorithm that is used for finding path from one node to another. So it can be compared with Breadth First Search, or Dijkstra's algorithm, or Depth First Search, or Best First Search. A* algorithm is widely used in graph search for being better in efficiency and accuracy, where graph pre-processing is not an option.

A* is a an specialization of Best First Search , in which the function of evaluation f is define in a particular way.

f(n) = g(n) + h(n) is the minimum cost since the initial node to the objectives conditioned to go thought node n.

g(n) is the minimum cost from the initial node to n.

h(n) is the minimum cost from n to the closest objective to n

Taking a square grid having obstacles and given a starting point and a target point. We want to reach the target point (if possible) from the starting point as quickly as possible.

An A* Search Algorithm, at each step picks the node according to a value-'**f**' which is a parameter equal to the sum of two other parameters – '**g**' and '**h**'. At each step it searches for the node having the lowest '**f**', and process it. '**g**' and '**h**' are defined as

g = the movement cost to move from the starting point to a given point on the plane, following the path generated to get there.
h = the estimated movement cost to move from that given point to the final destination. This is often called the heuristic, which is something like a smart guess. We can't know the actual distance until we find the path. There can be many ways to calculate this 'h'

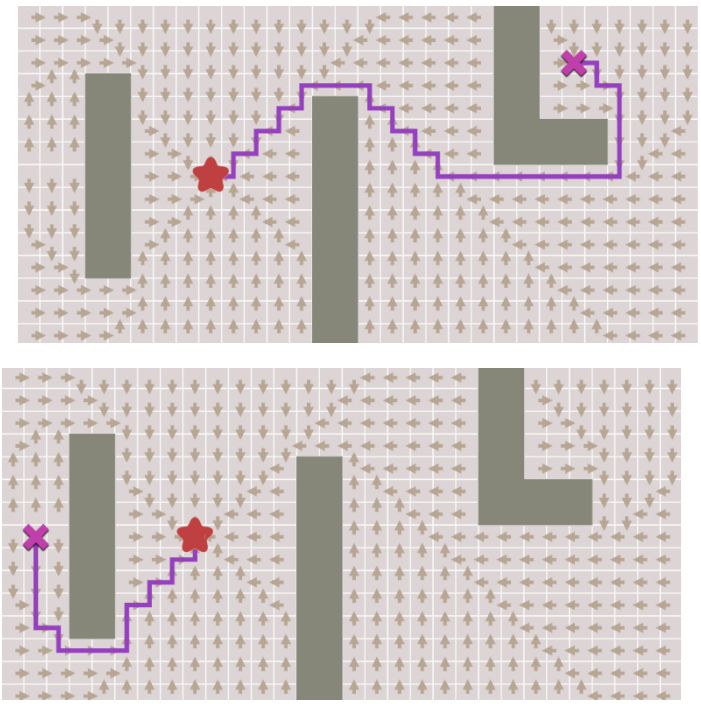
Exact Heuristics
We can find exact values of h, but that is generally very time consuming. Below are some of the methods to calculate the exact value of h.
1) Pre-compute the distance between each pair of cells before running the A* Search Algorithm.
2) If there are no blocked cells/obstacles then we can just find the exact value of h without any pre-computation using the distance formula/Euclidean Distance
Approximation Heuristics There are generally three approximation heuristics to calculate h
Manhattan Distance
It the sum of absolute values of differences in the goal's x and y coordinates and the current cell's x and y coordinates respectively, i.e.,
h = abs (current_cell.x – goal.x) + abs (current_cell.y – goal.y)
Diagonal Distance
It is the maximum of absolute values of differences in the goal's x and y coordinates and the current cell's x and y coordinates respectively, i.e.,

h = max { abs(current_cell.x – goal.x),abs(current_cell.y – goal.y) }

Euclidean Distance

It is the distance between the current cell and the goal cell using the distance formula

h = sqrt ((current_cell.x – goal.x)2 + (current_cell.y – goal.y)2)



The following is the algorithm for A* Pathfinding

1. Initialize open list
2. Initialize closed list put the starting node on the open list (you can leave its f at zero)
3. while the open list is not empty

a) find node with the smallest f on the open list, call it "q"

b) pop q from the open list

c) generate q's 8 successors and set their parents to q

d) for each successor

i) if successor is the goal, stop search successor.g = q.g + distance between successor and q successor.h = distance from goal to successor (This can be done using many ways, we will discuss three heuristics- Manhattan, Diagonal and Euclidean Heuristics) successor.f = successor.g + successor.h

ii) if a node with the same position as successor is in the OPEN list which has a lower f than successor, skip this successor

iii) if a node with the same position as successor is in the CLOSED list which has a lower f than successor, skip this successor otherwise, add the node to the open list end (for loop)

e) push q on the closed list end (while loop)

Contour Detection

Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same colour or intensity. The contours are a useful tool for shape analysis and object detection and recognition.

Contours of a binary image can be easily obtained by

```
im = cv.imread('test.jpg')
imgray = cv.cvtColor(im, cv.COLOR_BGR2GRAY)
ret, thresh = cv.threshold(imgray, 127, 255, 0)
im2, contours, hierarchy = cv.findContours(thresh,cv.RETR_TREE,cv.CHAIN_APPROX_SIMPLE)
```

For all the operations to be done using contours, OpenCV library has functions available for it.

The ball is being detected using a real time object detection system, You Only Look Once (YOLO). This system looks at the whole image at test time so its predictions are informed by global context in the image. It also makes predictions with a single network evaluation, unlike systems like R-CNN which require thousands for a single image. This makes it extremely fast, more than 1000x faster than R-CNN and 100x faster than Fast R-CNN.