**Report - Assignment 1** **Shabir Abdul Samadh**
COMP765 (Fall 2017) ID: 260723366
Prof. David Meger Date: 12/15/2017
Github URL[1]

# Localization using visual imagery

The objective of the assignment was to use visual image feedback from a camera *(in an underwater robot)* to localize the robot's position. The image received was a `400 x 400` image. In addition to the image, motion command feedbacks were also received to include in the localization system. Hence, in addition to the position estimate an orientation estimate was also to be include in the prediction to incorporate the robot motion.

# Implementation

## 0.1 Particle Filter

I have implemented a particle filter to estimate the robot-position in the given map. The particles are initialized randomly across the entire map *(which is a grid of* `800 x 2560`*)*. Each particle is assigned to have a random orientation at the beginning between $\pi$ and $-\pi$. The particles are also assigned some random noise *(motion-noise & sense-noise)* to incorporate the inaccuracies with the camera feedback as well as the motion commands. At every step in which the robot receives an `ImageCallback` *(which gives the latest image from the robot's camera)*, the received image is used to estimate it's current position.

The image is passed through a feature extraction algorithm[2] to detect the salient features in the image and to map them to best matching positions in the ground-truth *(the given ocean-floor map)*. This algorithm uses the `SURF` method for feature extraction and the `FLANN` method to match those features between the sensed image and the map-image. The output of this algorithm is the corner-points of the rectangle that localizes the sensed-image on the map. I have then used this output to calculate the `center-point` of this rectangular area which is taken as the location estimate for the robot. This estimate is taken based on the assumption that the camera is at the center of the robot.

I have also used an intermediary output of this algorithm *(which gives the Homography matrix for the camera)* to estimate the orientation `(yaw)` of the robot. The homography output from the feature-extraction algorithm is based on a system where the angle $\theta$ is calculated clockwise from `north`. However, the output of the `odometric` readings of the robot assumes the orientation angle $\theta$ to be measured anti-clockwise from the `east` direction. Hence, I have implemented a method (`convertYawFromCameraPoseToROSPlane()`) to translate the orientation output from the sensing-model to match that of the motion model.

The above process gives me a position estimate and an orientation estimate at every step an `ImageCallback` happens. Then I adjust the weights of all the particles based on these estimates. I first calculate the `euclidean-distance` between the current-position and the estimated position for each of the particles. Then I calculate the probability for this measurement *(for each particle)* from a `Gaussian` with `mean`($\mu$) **0** and `variance`($\sigma$) **50**. The variance measure

[1] https://github.com/Shabirmean/Assignment1
[2] The algorithm for feature extraction & position estimation was borrowed from the **OpenCV** documentation

was chosen upon several runs to find the radius around the estimated location from where the particles should have a higher probability. This probability measure is then normalized *(by the total probability)*. I also incorporate the orientation estimate into the particle-weights. I do this by calculating the difference between the estimated orientation of the robot and the particle-orientation. This difference measure is also translated to a probability-measure as explained before. The `variance`($\sigma$) for this `Gaussian` was set as 10% of the orientation range `(0 − π)` 0.314. This orientation-estimate-probability is multiplied with the normalized position-probability and is again normalized *(by the new total-probability)* to obtain the final weight of the particle.

Upon updating the weights of the particle, I re-sample the particles to further narrow down the location estimate of the robot. I have implemented the `Re-sampling Wheel` technique to give higher priority for particles with higher weights to be drawn.

### 0.2 Motion command modeling

At each step when the `MotionCallback` is triggered I update all the particles with the received motion command. The motion command contains the **forward-motion** to which I add some noise drawn from a `Gaussian` with a variance($\sigma$) of 0.1.This metric was chosen upon experimenting over different values over multiple runs. Some noise is also added to the intended orientation as given by the motion-command. The noise value for the yaw (orientation) is drawn again from the same `Gaussian` used for the orientation estimation in the filter above. Upon adding noise for the motion command, all the particles were updated to reflect the motion. Finally, the particle with the highest weight was chosen as the **best-estimate** for the robot's position and orientation.

## Learning process & outcomes

As a start to attempting this assignment I first learned the idea behind `Particle-Filters`. What I understood was the overall idea was to randomly choose a set of points in the state-space as possible estimates first and then iteratively re-sample to choose particles with higher probability. With this understanding, my approach was as if each particle was the robot itself. Upon receiving a visual feedback, I calculated the cumulative pixel difference between the received image and the image within the `400 x 400` square area around the particle itself *(as if the particle was in the middle and 200 pixels on both directions of both axises from the particle-position)*. I did this for each and every particle and used the difference-metric as the weight for that particle. However, this operation seemed very expensive and the program would not run at all, since I was calculating pixel-differences per **particle-surrounding** and had to separate out the `400 x 400` grid for each of them. Upon talking to some colleagues, I moved to the current implementation. This seemed counter-intuitive to my direct approach to particle filters. In the current implementation I obtain the estimate from the feature-extractor and calculate the distance to each-particle from this one-estimate to obtain a weight.

I also had to figure out a way to translate the orientation output from the feature-extraction process to match that of the **ros** system. Upon doing this a proper method had to be employed to use both position & orientation estimate to calculate the weights. Early, multiplication of the probabilities of these estimates would give a very small number between $0 - 1$. Hence, the approach described above was employed. In addition, I also had some trouble trying to reason with why I could not plot my estimates properly on the image. This had to do with mapping

the estimate-values to the proper pixel scales and also applying the motion commands to the particles under the right scale before re-scaling.

## Final Output

The estimates from the particle-filter seems to reasonably good when manually controlling the robot. However, the motion modeling seemed to be not very optimal for the given `ros-bags`. The robot is either faster than the estimation or is slower sometimes. It also seems that the program is slower during certain times since there are delays in the images being refreshed with the new estimate. I have tried varying the **motion-scaling-factor** with varying noise values to make it optimal.