Introduction to Robotics - Mini Project 1

Duarte Meneses Maria Duarte Vasco Silva

1 Introduction

We started by trying to understand what we were working with. For that we studied the references and tried to connect everything together.

After understanding how everything worked, we tried to start working with the packages, searching for how to use them and what parameters we could tweak in order to make the algorithms work well with our particular setup.

We could not show every result in this report so we arranged a Google Drive Folder with videos that we think are explicitly named. Next to each AMCL video there is a photo with the initial pose of the robot with the same name as the video.

2 Background

In this section we will present all the theoretical background needed to understand the topics present in this project. For each of the mobile robot challenges (localization and mapping) we will present the main algorithms used and establish a notation.

All information attained from a mobile robot sensor is **uncertain** because of noise and aliasing.

Bayesian Filters help us tackle this uncertainty problem as they allow us to estimate unknown probability density functions recursively over time using the readings from the robot's sensors and a statistical process model.

2.1 Localization

Localization is the problem of determining the robot's pose (position and orientation).

The robot we are using, the Turtlebot Waffle Pi 3, is a differential drive mobile robot. It only has encoders and a laser scanner so we will have to rely on those for localization.

2.1.1 Odometry Kinematic Model

Odometry is the use of motion sensors, in our case, encoders, to estimate the change of the robot's pose over time. In order to use the information from the encoders effectively we need to know the robot's kinematic model, that is, how we describe its motion over time. Knowing how much each wheel moved, Δs_r and Δs_l and the distance between the two wheels, b, we compute the robot's displacement Δs and the change in the pose by computing the changes of its different components x, y and θ:

$$\begin{aligned} \Delta s &= \frac{\Delta s_r - \Delta s_l}{2} \\ \Delta\theta &= \frac{\Delta s_r - \Delta s_l}{b} \\ \Delta x &= \Delta s \; cos(\theta + \Delta\theta/2) \\ \Delta y &= \Delta s \; sin(\theta + \Delta\theta/2) \end{aligned}$$

2.1.2 Odometry Uncertainty Model

Encoders, as all sensors and actuators, have error associated with their readings. One way of modeling this is described in [4]. This approach divides each movement in the time interval $(t-1, t]$, from $< \bar{x}_{t-1}, \bar{y}_{t-1}, \bar{\theta}_{t-1} >$ to $< \bar{x}_t, \bar{y}_t, \bar{\theta}_t >$ as a sequence of three steps. The first one, δ_{rot1} is a rotation in the position $(\bar{x}_{t-1}, \bar{y}_{t-1})$ so that the robot is aligned with its new position, $(\bar{x}_t, \bar{y}_t)$. The second one, δ_{trans} is a translation from $(\bar{x}_{t-1}, \bar{y}_{t-1})$ to $(\bar{x}_t, \bar{y}_t)$. The last one is another rotation, δ_{rot2} where the robot aligns itself with its new orientation, $\bar{\theta}_t$.

These deltas, can be computed in the following way (we will use some notation from the previous section instead of the book's notation to make it easier to understand how the kinematic model and the uncertainty model are connected in our case):

$$\begin{aligned} \delta_{trans} &= \Delta s \\ \delta_{rot1} &= atan2(\Delta y, \Delta x) - \bar{\theta}_{t-1} \\ \delta_{rot2} &= \Delta\theta - \delta_{rot1} \end{aligned}$$

[4] assumes that the "true" values of the translation and rotation are obtained from the measured ones by subtracting independent noise ε_b (Gaussian with zero mean and variance, b). The noise is proportional to how much the robot moved and to some alphas that we can adjust to express our trust in the odometry readings. In principle, the smaller these alphas are, the more we trust our odometry, since we're reducing the variance of our estimation. After accounting for the noise, we get:

$$\begin{aligned}
\hat{\delta}_{trans} &= \delta_{trans} - \varepsilon_{\alpha_3|\delta_{trans}|+\alpha_4|\delta_{rot1}+\delta_{rot2}|} \\
\hat{\delta}_{rot1} &= \delta_{rot1} - \varepsilon_{\alpha_1|\delta_{rot1}|+\alpha_2|\delta_{trans}|} \\
\hat{\delta}_{rot2} &= \delta_{rot2} - \varepsilon_{\alpha_1|\delta_{rot1}|+\alpha_1|\delta_{trans}|}
\end{aligned}$$

2.1.3 Model Matching and the Iterative Closest Point Algorithm

Model Matching is an absolute localization method, where the information from robot sensors is compared to a map. It matches sensor-based and world model maps, to determine translation and rotation to estimate the vehicle's absolute pose. The iterative closest point (ICP) is an algorithm used for model matching. ICP is used to minimize the difference between two clouds of points, one being the source point cloud and the other being the target point cloud. For each point in the source point cloud, it tries to find the closest point in the target point cloud (Euclidean distance). After, it calculates the rotation and translation between each pair, to have the transformation between the two clouds, then, it transforms the source point cloud to make it match the target point cloud, and it just keeps on iterating.

2.1.4 Extended Kalman Filters

In estimation theory, the extended Kalman filter (EKF) is the nonlinear version of the Kalman filter which linearizes a function at the current state estimate. In the EKF, the state transition and observation models don't need to be linear functions of the state but may instead be differentiable functions.

$$x_k = f\ (\ x_{k-1}, u_k\ )\ +\ w_k$$

$$z_k = h\ (\ x_k\ )\ +\ v_k$$

Here w_k and v_k are the process and observation noises which are both assumed to be zero mean multivariate Gaussian noises with covariance Q_k and R_k respectively. u_k is the control vector. The function f can be used to compute the predicted state from the previous estimate and similarly the function h can be used to compute the predicted measurement from the predicted state. The EKF has two phases: **Predict phase** - EKF is based on the Bayes probability which assumes the model and uses this model to predict the current state from the previous state. In this phase it predicts the state estimate and the covariance estimate. **Update phase** - Uses a measurement and it calculates de Kalman Gain, updates the state estimate and the covariance estimate.

A simple example of an application of EKF would be if we had a robot and we told it to move, when it finally stopped, we knowing the kinematic model , calculate a prediction for the state estimate, but we know that there is error associated with that (2.1.2). So, in order to get a better idea of where the robot really is, we use a measurement, to transform our current position prediction and turn it into a more accurate value. This measurement would correspond to an error between the predicted value of the previous step, and the actual measured present.

2.1.5 Monte Carlo Method

The last algorithm used in this first project is the Adaptive Monte Carlo Localization (AMCL). For this algorithm we will need a map of the environment but the robot does not need to know it's initial position. It is a technique to make predictions based on try-and-error. Not knowing it's initial pose, we describe the uncertain pose with many particles, also called samples. Each sample contains a possible robot poses which means that areas with more particles are more likely to be where the robot is. The output of the algorithm is the belief at a time, t, that is represented by a set of N particles:

$$Belief = \{x_t^1, x_t^2, x_t^3, ..., x_t^N\}$$

The procedure for the Monte Carlo Method consists in these five steps:

Initialization - Considering that the pose is unknown, N particles will be uniformly distributed on the map. Each of the particles having the same weight, $\frac{1}{N}$. If the robot's initial position is known the particles will be placed near that position.

Prediction - This step is based on the system model describing the motion of the robot. Each particle is moved as much as the amount of observed movement with odometry information and noise.

Update - On the update step, the probability of each particle is calculated and that probability is used to update every weight value.

Pose Estimation - Here we will use the position, orientation and weight of all particles in order to calculate three values: average weight, median value and maximum weight value. This values will be used on the next step.

Resampling - In this last step, the less weighted particles are removed and new ones will be generated close to the ones with the highest weights, always considering that the total number of particles, M, must be maintained. After the resampling we go back to the prediction step and we do it all over again.

2.2 Mapping

The basic difficulty is that the robot must know exactly where it is, so that it can update the right part of the map. However, to know where it is, the robot must already have a map [1]. One way to do it would be to use Particle Filtering to estimate $P(x_{1:t}, m|z_{1:t}, u_{1:t-1})$, that is, the probability of a given map and trajectory being the right ones, given all the commands (odometry) and all the observations. The problem with this approach is that while in Monte Carlo localization the search space was of dimension three (x, y, θ), here the state space depends on the size of the map. This can get quite big, and Particle Filtering is very inefficient for these cases. Gmapping actually uses Rao-Blackwellisation to factorize the previous probability in

$p(x_{1:t}|z_{1:t}, u_{1:t-1})p(m|x_{1:t}, z_{1:t})$. Now we can compute $p(x_{1:t}|z_{1:t}, u_{1:t-1})$ using a Particle Filter (and we are back in three dimensions) as it is very similar to the localization problem and then we can compute $p(m|x_{1:t}, z_{1:t})$ using a Kalman Filter.

3 Implementation

In this section we will describe the parameters that we can adjust within each of the localization packages so that we can relate the values we chose with our results.

3.1 EKF

The ekf_localization package allows us to tweak many parameters. The first ones are the alphas described in 2.1.2. Then we can tune some parameters related with the ICP algorithm: **max_correspondence_distance** is the maximum distance that two points can be apart and still be matched; the ICP algorithm can stop after it run for **max_iterations**, after two iterations generate a transformation whose difference is below **icp_optimization_epsilon** or after the some of Euclidean square errors is smaller than a user defined threshold, we think that **icp_score_scale** is related to that; ekf uses RANSAC for outlier detection to determine which points might be noise so, **ransac_iterations** is how much iterations the method should run for and **ransac_outlier_threshold** has to do with how selective we want the outlier detection to be; we can also adjust initial covariances.

3.2 AMCL

As the amcl package has many parameters, here we are going to focus on the ones that we actually changed: we can adjust the maximum and minimum number of particles with **max_particles** and **min_particles**; in this package it is not possible to not give an initial pose, we can change the variances, though, higher variance will generate particles further from the initial position; we can tune the alphas in 2.1.2; and we can change **recovery_alpha_slow** and **recovery_alpha_fast** so that we can have generation of random particles which are useful

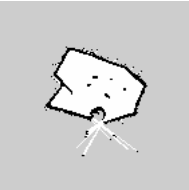

Figure 1: Map provided with the dataset

when recovering from an unknown position, for example, when "kidnapped".

4 Results

In this section we will present our results and discuss them. The map in the dataset can be seen in Figure 1.

4.1 EKF

At first, we tried to adjust the parameters related with the ICP algorithm. We tried more iterations and we tried a lower epsilon because we thought that the algorithm was generating pretty "weak" transformations and we wanted it to generate better ones. We also tried increasing the max_correspondence_distance in case the algorithm was being to strict when matching points and tried decreasing it in case it was matching points that shouldn't be matched. We also tried to tweak the RANSAC parameters in case the algorthm was discarding to many or to few points. Nothing seemed to work and as the odometry uncertainty increases over time we have to anchor to environment features to decrease that uncertainty, but since the ICP was not working properly, observing the environment was actually increasing the uncertainty. That's when we thought about seeing how good the odometry was, because, in case it was good, we could just decrease the alphas and the ekf_localization would start to consider more the odometry instead of the ICP, since when propagating the effects of actions, the variance would be low and we wouldn't need to rely on recognising environment features anymore. We placed the trajectory defined in the motion capture next to the one defined by the odometry readings. The result can be seen in Figure 2.

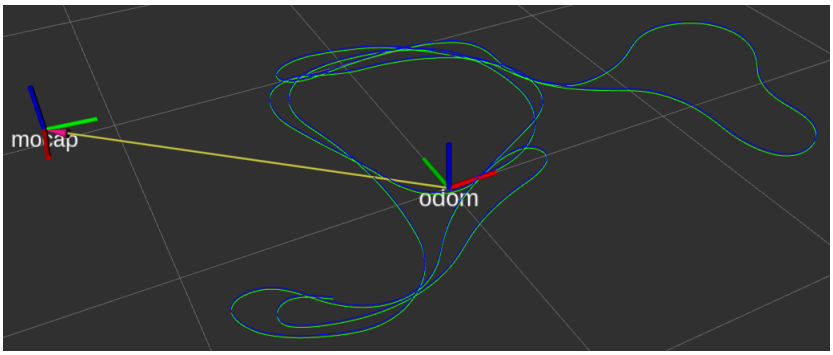

Figure 2: Odometry (green) vs Mocap (blue)

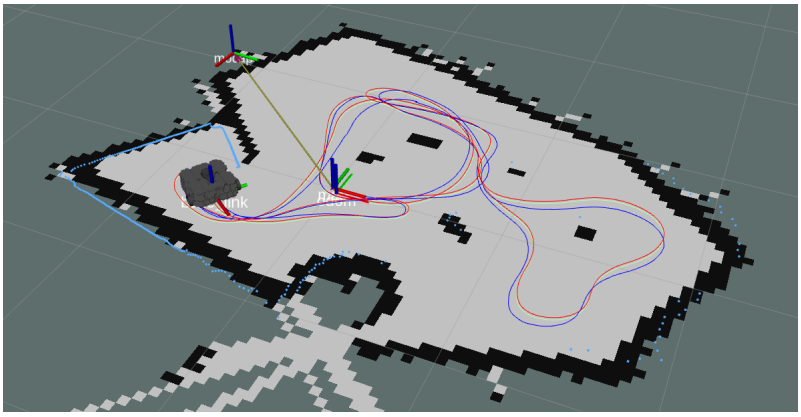

Figure 3: Odom(Green), Mocap(Blue), Estimated path(Red)

The odometry readings seemed pretty good, so we decreased the alphas, and the initial covariances and the ekf_localization started working very well. The estimated trajectory can be seen side by side with the captured one and the one defined by odometry in Figure 3.

4.2 GMapping

Figure 4 shows the map obtained with the gmapping package. We had some troubles with mapping the corridors when using the robots with clamps touching the floor. The contact with the floor might have made a difference when we consider how long the corridors are, and so we captured some maps that were not so good, as the one in Figure 5.

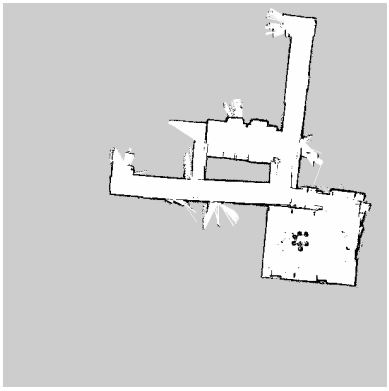

Figure 4: Accurate map

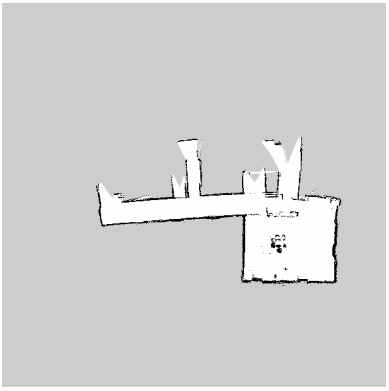

Figure 5: Strange map

4.3 AMCL

The amcl package was easier to adjust. Since we were not sure of how to measure the initial pose of the robot, we increased the initial covariance values a whole lot so that the particles generated at first would cover a larger array of possibilities instead of concentrating on the default pose value, (0, 0, 0). Because we were not giving amcl an accurate estimate of the initial pose, we wanted to generate random particles in case the initial ones didn't cover the position the robot was actually in, so we set recovery_alpha_slow and recovery_alpha_fast to the recommended values, 0.001 and 0.1 respectively. Since we captured a big map (we included the hallways), because we wanted to try the kidnapping and since we were lying to amcl about the robot's initial pose we increased the number of particles to be between 9000 and 1200 so that more random particles could be generated and more ground could be covered. In this case, contrary of what we did in ekf, we increased the alphas (0.5 for each), this was also related with the fact that we did not give the right initial pose and so we didn't want amcl to trust too much on the movement the robot was reporting.

We did try the kidnapping on the robot, but we stopped receiving laser scanner readings somewhere along the process and so, we did not get the expected results, though at certain point, the robot did self localize itself after being kidnapped. We started in the robot positioned outside the lab and we drove down the right corridor to the elevator hall until the particles converged. Once they did, we grabbed the robot and placed it in the corner of that same

corridor. At this point we would expect to see clouds of particles forming in the corners of both corridors, but we stopped receiving laser scanner readings and we got strange results since the robot started trusting the odometry too much. From there we drove the robot to the elevator hall again (where we would expect the cloud from the left corridor to disappear). There we started receiving readings again and the particles did converge to the real localization of the robot.

5 Conclusions

To sum up, on the EKF part of the project we were not able to make the ICP work properly so we compared the odometry with the ground truth data and found that it was pretty good. From there, we decreased the alphas, reducing the variance on the odometry errors, which means that the ekf_localization started trusting the odometry more. In the AMCL part, we were able to understand quickly that the recovery_alphas played a central role because we were not providing a trustworthy initial pose estimation. Regarding the kidnapping, we had some troubles with receiving laser scanner readings, but in the end, the robot did recover from the kidnap.

References

[1] Kevin P Murphy. "Bayesian map learning in dynamic environments". In: *Advances in Neural Information Processing Systems*. 2000, pp. 1015–1021.

[2] *ROS Robot Programming (English)*. ROBOTIS, Dec. 2017. ISBN: 9791196230715.

[3] Roland Siegwart. *Introduction to autonomous mobile robots*. Cambridge, Mass: MIT Press, 2004. ISBN: 0-262-19502-X.

[4] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT Press, 2005. ISBN: 0262201623 9780262201629.