Lab 1: SLAM for Karel in 1D

Ramón Júlvez, Ubaldo
696497
Cano Andrés, Lorenzo
736078

February 28, 2021

1 Introduction

The basic objectives defined for this first laboratory session are the following:

- Complete the implementation of a 1D SLAM mapping system algorithm based on the Kalman filter.
- Modify the implementation so that it creates many local maps instead of one global map.
- Compare the performance of the two approaches.

There are also two optional objectives that have also been completed:

- Depuration of the joined map with a Kalman filter approach.
- Finding an optimal local map size for the computational cost.

Unless otherwise specified, all the experiments presented were run with the default configurations for the robot motion and sensor range as precision given in the skeleton code, and with 1000 steps traversed by the robot.

2 Implementation of the basic 1D SLAM

A basic *skeleton* code has been provided. The following functionalities come already implemented in this base:

- The creation of the 1D environment where the Robot will be moving along with the features the robot will detect.
- The computation of the robot's movements, as well as the measurements of the features in the world.
- A map structure where to save the measurements of the environment features.
- Solving of the data association problem. Each feature in the environment has a unique ID, that is stored along with its position of the map.

Given all this, the only functionalities left to implement to have a basic SLAM algorithm are:

- Adding new features to the map.
- Updating of the map when an already-present feature is detected again.

The structure of the skeletal code allows the implementation of both of these characteristics by completing two functions.

2.1 Adding new features

In our set-up, the robot will take a fixed number of steps in a direction. The robot will be detecting a set of features for every step it takes. Some of the said features will have already been observed, while others will be new. In this section, we detail how we have proceeded to store the new features in the stochastic map.

In the first place, it is convenient to detail how the data of the stochastic map, and the measurements, is stored:

- The stochastic map is composed by:
  - x: The vector containing the features already in the map, being $x[1]$ the position of the robot (wrt. its original pose), and the rest the positions of the features (also wrt. the original pose of the robot). This vector is of size $m \times 1$.
  - P: the covariance matrix for all the features in the map. This matrix is if size $m \times m$
- And the measurements are composed by:
  - z: a vector containing the distances to the different features that the robot has detected from its current pose. This vector includes both new features and features already in the map. Being n the number of new feature measurements and f the number of "old" feature measurements, this vector is of size $(n + f) \times 1$
  - R: the covariance matrix for the measurements. The R matrix is of size $(n + f) \times (n + f)$.

To add the new features to the map, it is necessary to convert them, from a distance to the robot, to a distance from the reference, and to accordingly update the covariance matrix. This is done by means of a linear combination of the state vector of the features already on the map and the new measured features, as given by the formulae:

$$x = Ax + Bz \tag{1}$$

$$P = APA^T + BRB^T \tag{2}$$

The A matrix has two purposes. To isolate the changes to the features already on the map, and to expand the size of x so that the new features can be added. It also must add $x[1]$ (robot pose) to the new measurements (distance of features to the robot), so that the value stored on the map is the position of the features. Said matrix is constructed red as follows:

$$A_{(m+n)\times m} = \begin{bmatrix} \mathbf{I}_{m\times m} \\ \begin{bmatrix} \mathbf{1}_{n\times 1} & \mathbf{0}_{n\times(m-1)} \end{bmatrix} \end{bmatrix} \tag{3}$$

The B matrix needs to not alter the features already present in the map, and also, isolate the new features from the "old" ones, so that only the new ones are added to the map. Given that we know that the new features are at the end of the z vector, the B matrix is:

$$B_{(m+n)\times(n+f)} = \begin{bmatrix} \mathbf{0}_{m\times(n+f)} \\ \begin{bmatrix} \mathbf{0}_{n\times f} & \mathbf{I}_{n\times n} \end{bmatrix} \end{bmatrix} \tag{4}$$

2.2 Updating the map

As explained in the previous section, some of the observations that the robot makes on each step correspond to features already on the map. It is possible to use these new measurements to update the pose of both the robot, and the feature. To do this, we have made use of the Kalman filter.

Giving a detailed explanation of the Kalman filter is out of the scope of this assignment, but the overall process is described bellow:

$$\tilde{y}_k = z_k - H_k \hat{x}_{k|k-1} \tag{5}$$

$$S_k = H_k P_{k|k-1} H_k^T + R_k \tag{6}$$

$$K_k = P_{k|k-1} H_k^T S_k^{-1} \tag{7}$$

$$x_{k|k} = xk|k-1 + K_k \tilde{y}_k \tag{8}$$

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \tag{9}$$

In our case z_k is the the first f components of the z vector, $\hat{x}_{k|k-1}$ the x vector, $P_{k|k-1}$ the P matrix and R_k the R matrix, all of them explained in section 2.1. The only component left to be able to execute the Kalman filter is the H_k matrix.

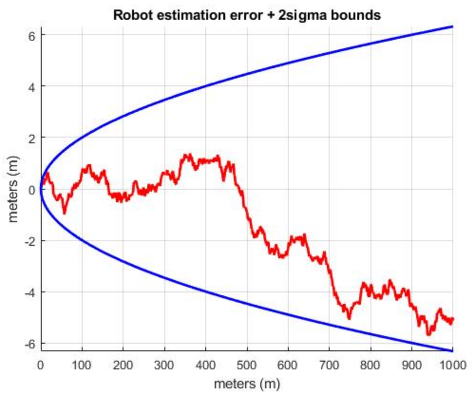
The z_k vector contains the measurements of the distance from the robot, to a set of features that are already on the map. The H_k matrix should then, return the expected values of those measurements from the pose of those features in the map. In our case, this means that it should subtract the current pose of the robot from the pose of the features. As the data association problem in our case is already solved, we know that the last f features of the stochastic map correspond to the features that have been measured again. With all this in mind, the H matrix ends up being:

$$H_{f,m} = \begin{bmatrix} -\mathbf{1}_{f\times 1} & \mathbf{0}_{f\times(m-1-f)} & \mathbf{I}_{f\times f} \end{bmatrix} \tag{10}$$

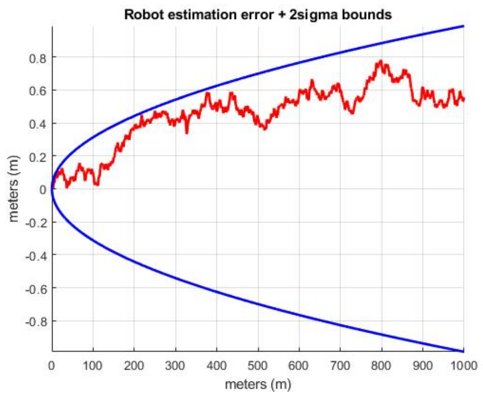
2.3 Results

In the fig.1 there is a comparison between the error in the estimation of the robot pose using pure odometry and using SLAM. In this figure one can observe that the uncertainty of the position of the robot is considerably smaller by using SLAM. From an uncertainty of the order of 6 meters, to an uncertainty of the order of 0.8 meters.

There is an interesting observation in fig.2. Looking at the error of the position of the features on the map, and the error of the position of the robot when it is near that feature (in fig.1b, there is a coupling of the errors of both the features and the robot position, but with a different sign.



(a) Before SLAM



(b) After SLAM. Sensor range of [0,2] meters.

Figure 1: In red, estimation error of all robot positions after each step. In blue, 95% confidence interval (2σ) for the estimation error.

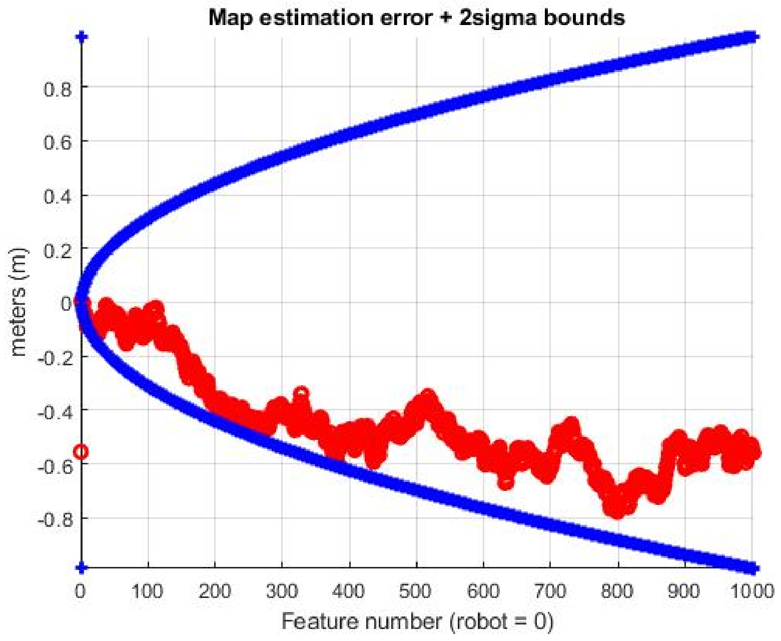


Figure 2: In red, estimation error of all the map features and the final position of the robot. In blue, 95% confidence interval (2σ) for the estimation error.

3 SLAM with local maps

One of the problems that the current approach has, is that the P matrix of the map increases size each time a new feature is added. As the Kalman filter computational cost increases with a quadratic dependency on the P matrix size, so does the time spent on computing each step (fig.3a).

3.1 Creating and joining maps

The solution for this problem is, instead of using a unique map for the complete environment, use a series of smaller sequential maps, and then join them together.

In our approach, we define a maximum size we want our local maps to have. At the end of a step, we then check if the size of our local map has exceeded the maximum size, and if so, we store that local map and generate a new empty one. Then, at the end of all steps, the global map is computed by joining all the sequential local maps.

The joining of the local maps follows the same process as the adding of new features to the local maps (sec.2.1). In this case, x is the global map of size m, and z the new local map of size n that has to be added to the global one. In this case, $x[1]$ is the pose of the robot wrt. the global reference frame when the local map was started, and $z[1]$ is the pose of the robot wrt. the local map, when the local map is finished.

The matrix A now has to maintain the features on the global map as they are, but it should shift the features in the local map to their position in the global reference (hence, adding $x[1]$). After joining, $x[1]$ should be the pose of the robot when finishing the local map, but wrt. the global map. The A matrix therefore results in:

$$A = \begin{bmatrix} \mathbf{I}_{m\times m} \\ \begin{bmatrix} \mathbf{1}_{n\times 1} & \mathbf{0}_{n\times(m-1)} \end{bmatrix} \end{bmatrix} \tag{11}$$

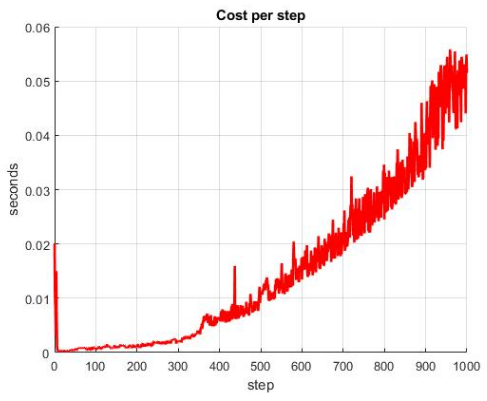
The B matrix now has to add $z[1]$ to $x[1]$ so the pose of the robot at the end of the global map is updated. It should not modify any of the features of the global map, and it should append the features of the local map at the end of the global map. Hence the B matrix is of the form:

$$B = \begin{bmatrix} \begin{bmatrix} 1 & \mathbf{0}_{n-1}^T \end{bmatrix} \\ \mathbf{0}_{(m-1)\times n} \\ \begin{bmatrix} \mathbf{0}_{(n-1)} & \mathbf{I}_{(n-1)\times(n-1)} \end{bmatrix} \end{bmatrix} \tag{12}$$

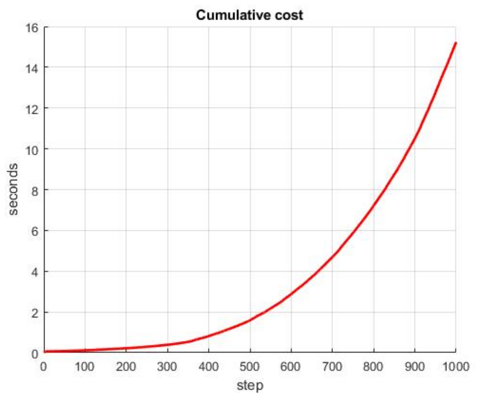
3.2 Performance

In fig.3 and fig.4 you can see the computational cost in seconds of executing the SLAM algorithm with the single map and local map approaches respectively. For the full map approach, we can see how the growth is exponential with respect to the step number since as we add more features to the map the matrices get bigger, and therefore the computations more expensive.

On the other hand, the cost when using multiple maps remains more or less constant and therefore the cumulative cost growth is linear since the maps don't get too large. Note that the spikes in the graphs correspond to dynamic memory management operations and thus have a big impact on the cost of a particular step but not on the cumulative cost.

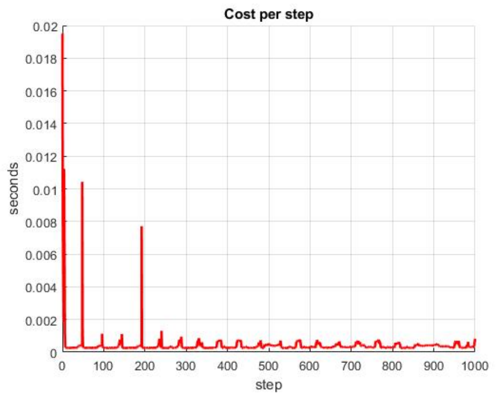


(a) Cost per step

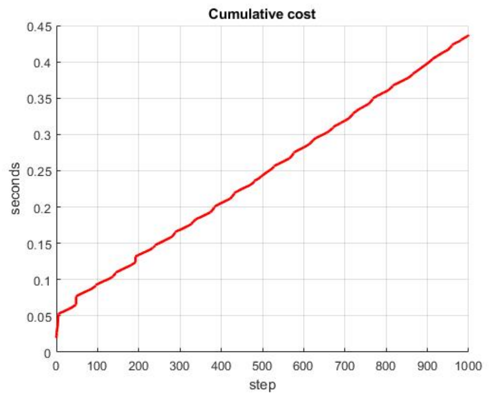


(b) Cumulative Cost

Figure 3: Computational cost for the execution of the SLAM algorithm with a single map for all the features in seconds.



(a) Cost per step



(b) Cumulative Cost

Figure 4: Cumulative cost for the execution of the SLAM algorithm with local maps of size 50.

4 Joined map depuration with Kalman filter

The above process works well for reconstructing a global map but has the problem of creating different entries in the state vector corresponding to the same real feature. This can be fixed by performing another Kalman filter update on the map, so as to fuse the repeated features. Consider x_i and x_j two entries on the state vector, which we know correspond with the same real feature on the map, as given by the data association. The values predicted for both will be different as they come from different measurements, but ideally, we would want them to be equal. We can achieve this by means of "making up" a new measurement and updating the map with the Kalman filter.

We will consider our measurement as the difference of the true values of both entries (which we know will be zero since they represent the same feature):

$$z = x_i - x_j = 0 \tag{13}$$

We also consider the measurement error to be zero ($R = 0$) since we can say the above is true with certainty. With this we can perform a Kalman filter update with this new measurement, all that is left is to define the H_k matrix. Since we know that :

$$z_k = H_k x_k \tag{14}$$

Given f the number of features with repeated entries and m the total number of features, H_k will be a matrix size $fxm+1$, where each row will have a 1 on column i and a -1 on column k.

On fig.5 and fig.6 you can see the reconstructed maps before and after depuration respectively, for 100 steps and a sensor range of 4, which makes the improvement more noticeable. Note how on the depurated maps the results are very similar to those obtained by the full map implementation.

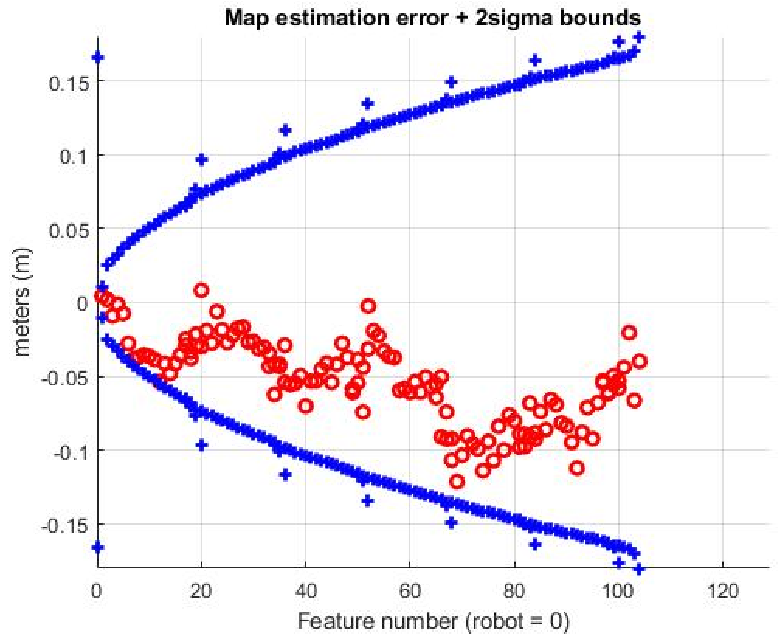


Figure 5: In red, estimation error of all the map features and the final position of the robot, with linearly joined maps. In blue, 95% confidence interval (2σ) for the estimation error.

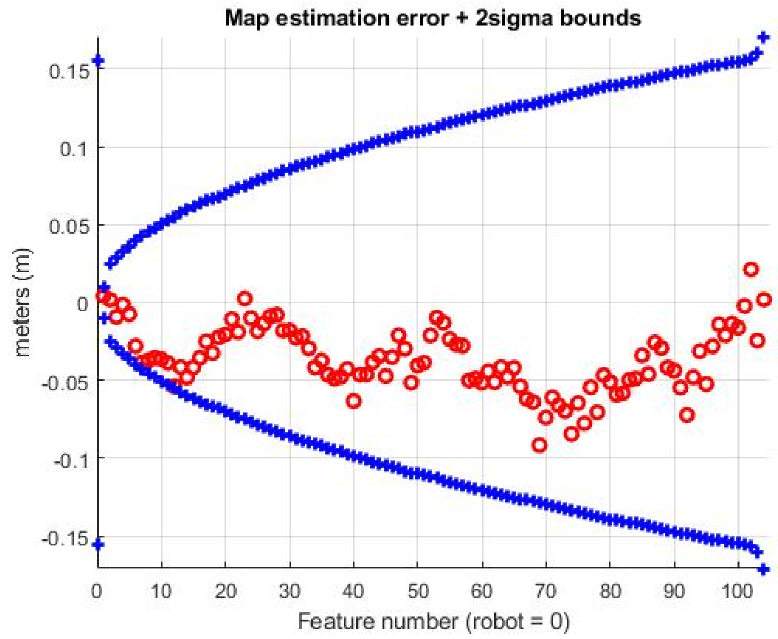


Figure 6: In red, estimation error of all the map features and the final position of the robot, after map depuration. In blue, 95% confidence interval (2σ) for the estimation error.

5 Selection of optimal size for local maps

Though we have already shown how utilizing local maps of a limited size considerably improves performance in comparison to using a single map, the optimal size of the local maps p so as to minimize the overall cost of computation is not trivial to calculate. For this objective, we implement a naive approach based on an exhaustive search and implement the equations described [1] so as to calculate the theoretical optimal value.

5.1 Exhaustive search

For calculating the experimental optimal value of the local map size, we run and measure the SLAM algorithm for values between [5,1000] with an increment of 5. For each local map size, the algorithm was run 10 times, so as to calculate the mean between experiments and obtain a more accurate result.

These results can be seen in fig.7. It can be observed how for increasingly big maps the time tends to increase, but also for very small maps (size under 20) as the reduced time for the calculations does not compensate for the increased amount of maps required.

The optimal value obtained experimentally for the local map size was thus $p = 80$. The speedup of using the optimal size with respect to using the full map is around 26 times. Note how this result is only valid for these exact parameters of the robot exploration and environment, and the search would have to be repeated if any of this change. Also, note that the exact times here are different from the ones reported above, this is simply due to this experiment being run on a different computer.

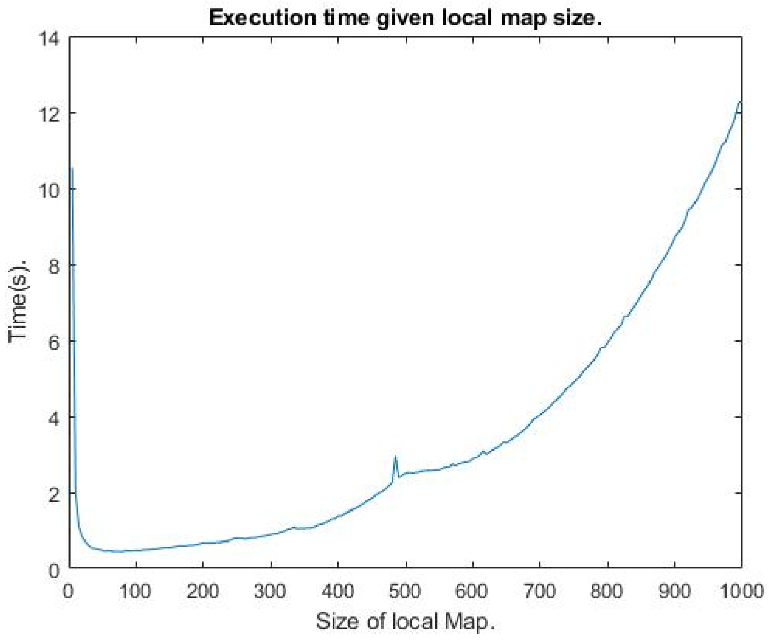


Figure 7: Execution time of the SLAM algorithm for a varying size of the local maps.

5.2 Theoretical cost

We also perform a theoretical calculation of the number of operations required to execute the SLAM algorithm for local map size, p, given the total number of features on the environment,

$P = 1000$, the number of re-observed features on each step, $r = 1$, and the new features observed on each step, $m = 2$. The result of plotting this for all the values of $p \in [5, 1000]$ can be seen in fig.8.

Overall, the results are very similar to the experimentally calculated ones as seen in fig.7, though these can't be directly compared since for the experimental results we can only measure time and not directly the number of operations. The optimal theoretical value for the local map size obtained is p = 79, which coincides with the experimental result of 80 (note that since the increment size is 5 this is the closest tested value).

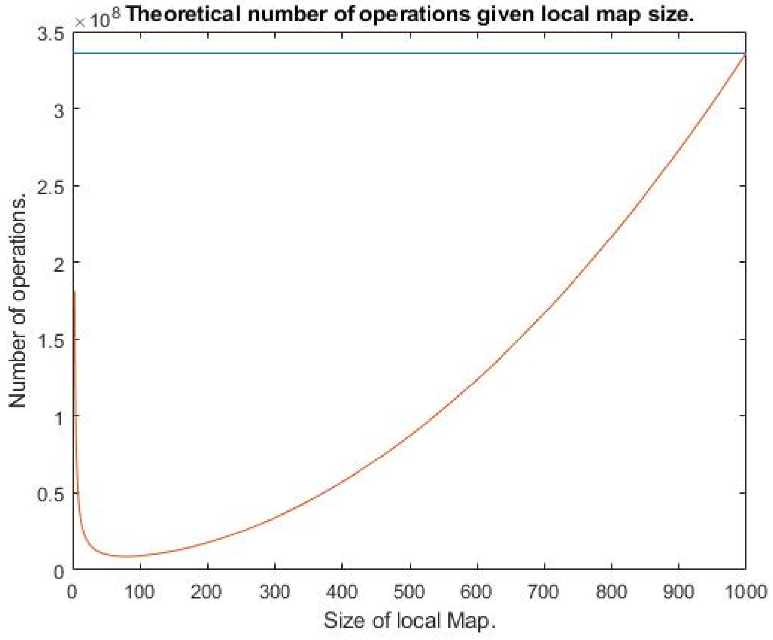


Figure 8: In orange, theoretical number of operations required given the local size of the maps. In blue number of operations required for considering the full map (for comparison).

References

[1] L. M. Paz and J. Neira. "Optimal local map size for EKF-based SLAM". In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2006, pp. 5019–5025. DOI: 10.1109/IROS.2006.282529.