[S5] - FastSLAM using UWB

João Canas, *MEAer, IST,* Manuel Rosa, *MEAer, IST,* Pedro Trindade, *MEAer, IST,* and Pedro Carmo, *MEEC, IST*

***Abstract*—This article describes the work done within an academic project for the curricular unit of Autonomous Systems at Técnico Lisboa. The project had as objective, the implementation of the FastSLAM algorithm using odometry readings, from an ActivMedia Pioneer 3DX robot, and ultra wideband distance measurments, from Kio Indoor Positioning System.**

***Index Terms*—FastSLAM, UWB, Autonomous Systems**

I. Introduction and Motivation

THE problem of simultaneous location and mapping (SLAM) is currently a topic subjected to a lot of investigation. It is particularly interesting because it is a more generic approach than considering localization and mapping separately since in a new environment everything is unknown at first.

The principle of SLAM consists in using a relative localization method to estimate the position and an observation model to map the environment in relation to the agent's location. The traditional approach to this problem consists of implementing an extended Kalman filter (EKF) to estimate a location and a map, however, this approach has some problems namely growing quadratically with the complexity of the map. A more recent method is FastSLAM, which represents a stochastic approach, where a particle filter is implemented to estimate the most likely state. This new approach has been proven to present several advantages over EKF SLAM such as a smaller complexity growth and is the one considered in this project.

The observations given to the SLAM algorithm may come from a variety of sensors commonly used in mapping problems. The one most widely implemented is the laser scanner which can provide both a distance and a bearing for each measurement. In this work, an alternative sensor for observations was studied, the ultra wideband (UWB) range-finder. In recent years, UWB systems used in communications have been adapted to and implemented in localization systems. These systems measure the distance between a tag and a set of anchors and are usually implemented to localize a tag knowing the precise location of all the anchors. This project intends to study the applicability of such sensors to the mapping step of SLAM where the problem is reversed and the positions to be calculated are the ones of the anchors.

II. Methods and Algorithm

In this section, we will briefly describe the methods considered. Algorithms used will also be referred, as well as ROS packages and Python libraries, as they are all relevant for implementation

In our project, FastSLAM was used to solve the problem of simultaneous location and mapping. This solution implements a stochastic approach using the algorithm of a particle filter. For each particle the joint distribution of a pose and a landmark given the evidences (observations and movements) is factorized, based on Rao-Blackwellization, and re-written as the product of the pose probability given the evidences (path posterior) and the map probability given the position and the observations (map posterior).

$$p(x_{0:t}, l_{1:M}|z_{1:t}, u_{1:t}) = p(x_{0:t}|z_{1:t}, u_{1:t})p(l_{1:M}|x_{0:t}, z_{1:t})$$

Since all landmarks are conditionally independent knowing the pose, the map probability is the product of all individual landmark probabilities. An EKF (extended Kalman Filter) is then used for the representation of the landmarks, this way, each particle has one EKF per landmark.

Each particle has a representation of a possible robot pose. In our case, we consider that the robot only moves in 2D and its pose can be represented as:

$$p_{p_i} = [x_{p_i} y_{p_i} z_{p_i} \theta_{p_i}]^T$$

, where θ_{p_i} represents the orientation of the ith particle, and z_{p_i} is constant. For simplification we consider a frame where $z_{p_i} = 0$ without loss of generalization. Each particle also has a map with the position of the landmarks. Each landmark is positioned in a 3D plane and can be represented as:

$$p_{L_j} = [x_{L_j} y_{L_j} z_{L_j}]^T$$

, where $z_{L_j} \geq 0$ since all landmarks are positioned above the robot.

Correct initialization of the FastSLAM algorithm is very important and a non-trivial problem. In a location only problem each particle of the filter is typically initialized in an pose sampled from an uniformed distribution over all the environment space. However, in a FastSLAM problem each produced map is drawn with respect to its particle pose. For this reason, the randomness introduced initially would never be resolved and the difference between particle's maps would have a component due to their different initializations. Therefore, all particles are initialized in the same position, corresponding to the origin of the referential, and with the same orientation, point along x-axis.

Every time a new landmark is observed, an estimator for its position must be initialized and added to the map. The obvious solution is to initialize the landmark in the same position as the particle with a large co-variance and wait for the EKF to move it to its correct position. This solution works in situations with good observability, however, performing this initialization in this case means that we expect the filter estimation to converge to the accurate position in a 3-dimensional space with only one distance measurement to this landmark and no initial information, which will not occur. To obtain an initial guess with useful information a trilateration algorithm, based

in least squares estimate, is implemented and invoked every time a new landmark is found.

Figure 1 represents the algorithm implemented and highlights the importance of initialization.

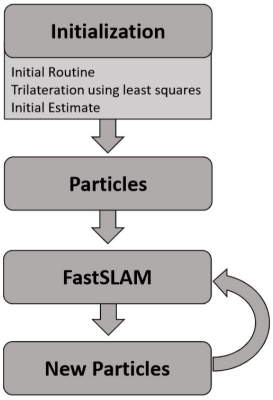


Fig. 1: Algorithm Diagram

To implement all the referred methods some libraries from ROS and python were imported and used. Considering ROS, ROSPY was used as an interface with ROS and python, which allowed to get information from the ROSARIA topic *pose* (that returns the pose of the robot based on its odometry) and from the UWB tags, about the distances from them to each anchor. Besides this, ROSPY was also used to publish some results about the calculated estimation about the trilateration results and the FastSLAM implementation results about each particle pose and anchors position, for visualization purposes. Regarding tf, it was used to define the odom frame (fixed frame) in RVIZ, to later use the UWB measurements to be published within the base_link frame and be transformed into odom frame. Other Python libraries should also be considered, such as numpy, math and scipy.stats, used to implement the algebrical, mathematical and statistical operations, respectevily. Also, there were some imported message types, that were required to import in order to assure compatibility on the used topics.

III. Implementation

A. Trilateration

To have a proper initial input for the Kalman filter, an initial guess of the position of the particles is obtained using a trilateration algorithm. This algorithm receives P positions of the robot, obtained from the robot odometry $(x_i, y_i, 0)$ and the distances d_i measured by the UWB tag to each landmark in those positions. Then, a set of points (x_j, y_j, z_j) in the free space are evaluated using the following cost function:

$$E_j = \sum_{i=1}^{P} \left(d_i - \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + z_j^2} \right)^2 \quad (1)$$

The algorithm then selects the point that minimizes this function and generates a new set of points around it and the process is repeated in order to get a better approximation of the position of the landmark, according to the used measurements. Since all the positions used are on the ground plane, there will always be an ambiguity with this algorithm. To avoid this, all the points considered for the algorithm are generated with a positive z coordinate to avoid a solution below the robot.

The number of points generated and the number of iterations the algorithm does are parameters that were adjusted to achieve a good trade off between accuracy of the result and running time.

This initialization is ran whenever the tag detects a new landmark.

B. Particle Filter

In sequence to what was introduced in the previous section, we will provide in a bigger detail, how the particle filter was implemented.

Firstly, we began by generating an initial set of N particles. We decided to initialize the particles with the initial position of the robot, in the odometry frame, which is the origin, and orientation equal to zero.

Now that our particle filter is initialized, lets explain in detail how it proceeds to the update. In each update step, the particle filter receives an input from odometry, the consists in the previous pose and actual pose of the robot, and makes the difference between these, to obtain what the robot moved in this step. After this, for each particle, we add this difference, with some random noise, so that each particle gets a different input. With this done, we get a new pose for the particle.

We then proceed with the weighting. To get each particle weight, we use the landmarks estimated position, and the particle pose, to compute the distance we should measure, and then compare it with the measurement we get from the UWB. We do this for both tags, which, assuming the measurements are independent, results in the product of $2M$ probability densities, being M the number of observed landmarks. To estimate this measurements we first needed to compute the position of the tags based on the pose of the particle. This can be done based on the scheme represented in figure 2

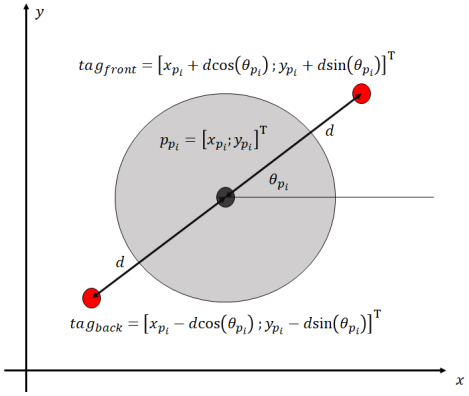


Fig. 2: Tags positions computation

As we can see, since we consider the particles are at the height of the tags, and we consider that height to be equal to zero, we can easily get the position of the tags based on the

pose of the particle, using the expressions on figure 2, that are expressed below:

$$p_{tag_{front}} = \left[x_{p_i} + d\cos(\theta_{p_i});\ y_{p_i} + d\sin(\theta_{p_i}); 0\right] \quad (2)$$

$$p_{tag_{back}} = \left[x_{p_i} - d\cos(\theta_{p_i});\ y_{p_i} - d\sin(\theta_{p_i}); 0\right] \quad (3)$$

Having now these positions, and computing their distance to the estimated position of the landmarks, we get a set of distances, $\hat{d_{jk}}$, each one associated with a combination of observed landmark j and tag k, which we can compare to the real measured distances, to get the weights, by doing:

$$w_i = \prod_{k=1}^{2}\prod_{j=1}^{M} p^*(D = \widehat{d_{jk}}) \quad (4)$$

In this equation, p^* represents the value of the probability density function of the random variable D, which was chosen to be a normal distribution of mean d_{jk}, which is the value of the measured distance from landmark j to tag k, and variance σ_d^2, which can be adjusted to obtain better results.

After getting our belief represented by a set of weighted random samples, we normalize the weights. Then to decide if resampling is necessary, we compute the number of effective particles and check if it is smaller than a given threshold $(N/2)$ and resample if it is. Resampling is done by drawing N random samples from the set of samples, with probability of drawing a given sample is given by its weight, w_i.

C. Kalman Filter

As explained before, the FastSLAM algorithm usually uses the EKF (extended Kalman Filter), one for each landmark. This uses the distance measurements from the front tag to the landmark (we are using the back tag only to correct the orientation), obtained from the UWB, to estimate the position of the landmarks. However, this is not feasible with a standard approach, since the UWB only gives us a distance, and for our model, we would have observability problems, and the EKF wouldn't converge. That being said, we proceeded with a different approach, based on [1].

First, we began by writing our observation as:

$$o_{L_j} = \begin{bmatrix} x_{p_i} + d\cos(\theta_{p_i}) \\ y_{p_i} + d\sin(\theta_{p_i}) \\ z_{p_i} \end{bmatrix} + \begin{bmatrix} d_j\sin(\phi_j)\cos(\alpha_j) \\ d_j\sin(\phi_j)\sin(\alpha_j) \\ d_j\cos(\phi_j) \end{bmatrix} \quad (5)$$

In this expression, ϕ_j represents the angle between the Z axis and the line the passes through the jth landmark and the particle, and α_j represents the angle between the projection of this line in the XY plane and the X axis.

We also need to write the covariance matrix for our observation, having in mind that our observation measures only the distance, and that is the direction in which we have smaller covariance. With this said, and transforming coordinates for the covariance:

$$\Sigma_{r\phi\alpha} = \begin{bmatrix} \sigma_r^2 & 0 & 0 \\ 0 & \sigma_\alpha^2 & 0 \\ 0 & 0 & \sigma_\phi^2 \end{bmatrix} \quad (6)$$

$$\Sigma_{XYZ} = R_y(\pi - \phi)R_z(\alpha)\Sigma_{r\phi\alpha}(R_y(\pi - \phi)R_z(\alpha))^T \quad (7)$$

Where $R_y()$ and $R_z()$ represent the rotation matrices around the Y and Z axis respectively. We chose σ_α^2 and σ_ϕ^2 to be twenty times greater than σ_r^2. This forces the filter to correct mostly in the direction he supposes it is taking taking the measurement, and not discard completely the information in the other directions.

Since we don't have access to this angles, we use the predicted landmark position to obtain an estimate of these, assuming our prediction is approximately right. These are obtained by doing:

$$\hat{d_j} = \sqrt{(x_{\hat{L_j}} - x_{p_i})^2(y_{\hat{L_j}} - y_{p_i})^2 + (z_{\hat{L_j}} - z_{p_i})^2}$$

$$\hat{\phi} = \arccos\left(\frac{z_{\hat{L_j}}}{\hat{d_j}}\right)$$

$$\hat{\alpha} = \arctan\left(\frac{y_{\hat{L_j}} - y_{p_i}}{x_{\hat{L_j}} - x_{p_i}}\right)$$

This way, our model is quite simple to write and, since we don't have the need to linearise it, in our case the EKF is actually a standard Kalman Filter, and the model matrices are:

$$F = I$$

$$B = 0 \implies \text{No control input}$$

$$H = I$$

In which I represents the identity matrix, of adequate dimensions (3x3 in this case). Still, this method takes a while to converge, and in order for improving it, we needed to get a good estimate of the initial landmark positions.

D. ROS

For this implementation, it's used ROSPY which is a library that allows its users to take advantage from the ROS framework using PYTHON which is an high level programming language, both of them providing high levels of abstraction for implementation purposes.

ROS works by means of subscribe and publish messaging pattern, which allows the transmission of data into topics without programming it for a specific receiver. By creating a node, one can publish and subscribe topics. The vital data for the algorithm implementation of this project comes from three subscribers: the topic that returns its pose, published by ROSARIA and the ones that return the distances from the tags to each anchor, one for the front tag and another for the back tag. This last two required the installation of their drivers, which, when running the given launch file, automatically publish their respective values on their own topics. Each of this subscription has a callback function that is called when data is received, being assigned to them in that context. The data received comes in a specified format, requiring the importation of *uwb_anchor_array* from *monarch_uwb.msg* for the tags and *Odometry* from *nav_msgs.msg* for the pose.

Regarding publishing topics, it was used *PoseArray* and *Pose* messages, from *geometry_msgs.msg* to publish readable messages about particles pose for RVIZ, as well as *Marker* and *MarkerArray* from *visualization_msgs.msg* to publish readable messages for landmarks positioning in RVIZ. TF library was

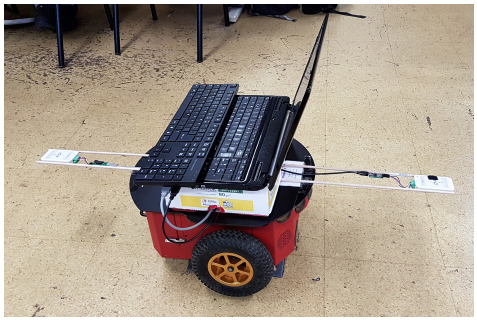


Fig. 3: Pioneer and UWB tags configuration

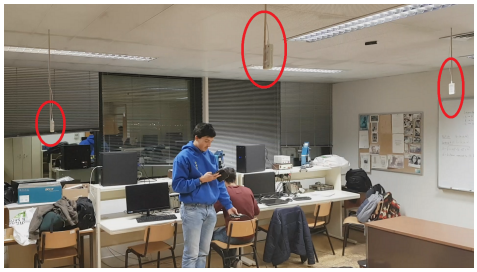
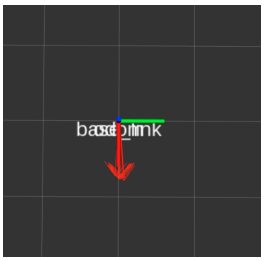
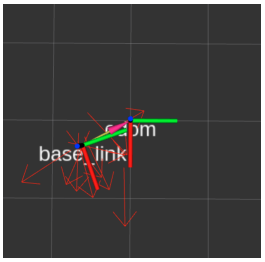


Fig. 4: Landmark positioning for experimentation



(a) No landmarks (1sec)



(b) No landmarks (170sec)

Fig. 5: Algorithm without landmark identification

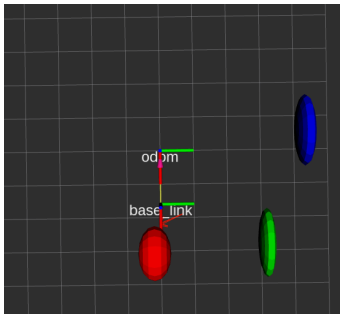


Fig. 6: Algorithm with imprecise landmark initialization

also relevant to read the transformations from *base_link* to *odom*, thus providing a fixed frame *odom* on which all the provided and calculated data is transformed to, assuring compatibility between every component of the visualization on RVIZ.

IV. Experimental Results

The tests for the algorithm explained in the previous sections were done in the Autonomous Systems laboratory in IST using one of the Pionner robots available. Two UWB tags were installed on the robot as it can be seen in figure 3 and three UWB anchors were hung on the ceiling to avoid interference from the environment as shown in figure 4.

Then the robot was driven trough the laboratory in random paths while taking readings from odometry and the UWB tags. In some of the tests, the exact position of the UWB anchors was measured as well as a few positions of the robot. These measurements will allow to evaluate the error given by the FastSLAM algorithm in the future.

The first check made to ensure the partial functioning of the algorithm was running it with real data but without doing landmark association. In this case, there is no landmark to evaluate each particle's position estimated from the odometry (with some added noise) and all particles are always equally likely. Results from this test were expected to show an initial pose of particles in accordance with the base_link in the odometry referential which would gradually disperse around the odomoetry readings. Figure 5 is taken from RViz and shows the particles at start, $t = 0$s, and at end, $t = 170$s, when the robot had returned to the initial pose. Such results are in accordance with the expected.

The second test performed was to check if the filter didn't diverge given it was well initialized. To test this, the robot initial pose and the landmarks positions were measured and given as inputs. The results proved, given the precise initialization, that the filter didn't diverge and the final estimate was in accordance with the real positions of the landmarks as well as the real pose of the robot.

If the landmarks position is not initialized precisely, the algorithm will not convert. Figure 6 shows this exact situation, the landmark represented in blue is too far from its real position which leads to an improper estimation of the robots pose by the particle

Finally the algorithm was tested with trilateration as a method for initializing each new landmark. Since we only have 3 landmarks it was only run once at the beginning of the program. The estimate of the landmark was not very precise but it was enough to lead the fastSLAM algorithm to conversion. At the present moment precise data analysis and error comparison is still in process.

In further work data recorded will be processed and commented in detail. Since this is a stochastic process, one simulation is not enough and an average between runs in simillar conditions should be taken.

V. Conclusion

With this project it was possible to implement a FastSLAM algorithm for localization of a robot in an unknown environment using UWB readings.

The main conclusion that can be drawn regards the observability of this type of systems. Using only the measured distance to the anchors proved to be a challenge to the development of the algorithm. It was possible to conclude that to estimate a position in space, which consists in three independent states, the bearing of the said position when observed by the robot is also important. Thus, a redesign of

the observation model used in the Kalman filter was required. Although this redesign provided us with better results, it still doesn't work very nicely, mostly due to the difficulty in estimating the z component of the landmarks position.

It was also concluded that communication systems such as the Ultra-Wide Band may give very noisy measurements. The multipath effects required the anchors to be placed in placed far away from reflecting surfaces such as the ground (which may have cables running below it), the metal ceiling of the laboratory and the electrical equipment present in it. It was also important to place the tags away from each other to reduce the effects of the noise in the orientation estimation.

For further work on this topic it is suggested that the system is tested in an environment with more anchors and more space between them. This would allow to evaluate the viability of this method over longer distances and in situations where the robot cannot receive measurements from the same UWB anchors during the entire mission.

REFERENCES

[1] G. Kantor and S. Singh, "Preliminary Results in Range–Only Localization and Mapping", in *Proceedings of the IEEE Conference on Robotics and Automation (ICRA '02)*, 2002, Vol. 2, pp. 1818 - 1823

[2] M. Montemerlo, S. Thrun, D. Roller, and B. Wegbreit. "FastSLAM: A factored solution to the simultaneous localization and mapping problem", in *Proc. AAA1 Nat. Conf. Artif. Intell.* , 2002, pp. 593–598

[3] P. Lima. SAut, Class Lecture, Topic: "SIMULTANEOUS LOCALIZATION AND MAPPING (SLAM)" Instituto Superior Técnico, 2017

[4] S. Thrun, W. Burgard and D. Fox. *Probabilistic Robotics*. Cambridge, Massachusetts: MIT Press, 2006, pp.

[5] W. Hereman and W. S. Murphy "Determination of a Position in Three Dimensions using Trilateration and Approximate Distances" Colorado School of Mines, 1995