Distributed unicycle system for the estimation of the 3D position of a target

Marco Sterlini

Abstract - This study aims to estimate the 3D position of a target by employing distributed estimation utilizing a set of unicycles deployed in an indoor environment. Each unicycle is equipped with a UWB tag essential for determining its own position and orientation, complemented by a 2D camera used for target localization and navigation. Upon the proximity of all unicycles to the target, a distributed and scalable estimation of the target's three coordinates is executed.

I. INTRODUCTION

The idea is to provide a solution to the problem of the estimation of the position of a target without prior knowledge. To do so the whole study was portrayed with the aid of simulations in GAZEBO using the ROS infrastructure to manage the communication between the different modules of a single unicycle and the communication between different agents. The first problem to solve was the estimation of the 2D position and the orientation of every single unicycle. To do so the idea was to exploit the principle portrayed in [1] with a trilateration based approach implementing UWB tags, one in each unicycle and others in known and fixed positions in the room's roof. The second problem to solve was the recognition of the target, a yellow sphere located on top of a stand was used and each unicycle was given computer vision algorithms to properly analyze the images received from their own camera and identify the center of the sphere in their own camera reference system. Finally, once every agent is in possession of the information regarding their own position and orientation the last problem to address was the distributed estimation of the target where each agent communicated its own state along with the useful coordinates of the target in each own camera reference system.

II. ADOPTED MODELS

Given the main practical focus of the project some approximation were portrayed in the design of the communication system, hence it was assumed that each agent of the system was able to communicate with everyone at every given moment. It's important to stress nevertheless the distributed nature of the architecture.

A. Communication System

In the current model, the communication system assumes an idealized environment where each unicycle can communicate freely with every other unicycle without any message loss.

The communication protocol implemented is the ROS one, really similar to the industrial protocol MQTT based on topics with publisher and subscribers.

Each robot serves as a node in the network and participates in the communication independently without a central point of control or coordination. The communication follows a peer-to-peer model since each robot can communicate directly with others robots in the network. The communication protocol allows to exchange information in an asynchronous way that allows flexibility and scalability.

This communication architecture was used both for the inter-robot communication and the communication between modules of a same agent. In a real case scenario this would not be necessary but since the simulation was developed in many different modules, this approach was considered the cleanest one. The difference between a external and internal communication will be established by the hierarchy of the topic of interest. As an example a topic "/$robot1/data$" will be considered as a internal message between modules of "robot1", a message in a topic called "/$localization\_data$" will be considered a public topic where every robot can take information from.

For a real-world implementation, Ultra-Wideband (UWB) transceivers, such as the DecaWave DW1000 used in [1], in conjunction with the ROS communication framework and a computing unit like a Raspberry Pi running it would be employed, all the communication between sensors and the system would then happen via serial communication. These technologies offer high precision and robustness but still face practical challenges such as line-of-sight (LoS) requirements, packet loss, and network topology constraints.

Line of Sight (LoS): UWB communication generally requires a clear line of sight for optimal performance. In environments with obstacles or varying terrain, maintaining LoS can be challenging, potentially leading to signal degradation or loss. In the particular application of a indoor environment however this will not have a strong impact.

Packet Loss: Even with UWB, packet loss can occur due to interference, multipath effects, or occlusions. This impacts the reliability of data exchange, potentially resulting in incomplete or outdated information being shared between unicycles.

Communication Graphs: The system can be represented as a communication graph where nodes (unicycles) are connected by edges representing UWB communication links. In reality, this graph might be sparse or dynamic due to the previously mentioned issues, as opposed to the idealized fully connected graph implemented in this project.

B.System Model

Regarding the system model each robot was considered as a unicycle. Hence the discretized model of the state was used:

$$\begin{aligned} x_{k+1} &= x_k + \Delta t \cdot (v_k \cdot cos(\theta_k)) \\ y_{k+1} &= y_k + \Delta t \cdot (v_k \cdot sin(\theta_k)) \\ \theta_{k+1} &= \theta_k + \Delta t \cdot \omega_k \end{aligned} \tag{1}$$

C.Problem Formulation

The model used for this project was the unicycle, hence the state estimation problem consisted in determining the $x-y$ position in a plane and the orientation θ.

- Self state estimation:
  A number m of UWB tags were deployed in GAZEBO with customly chosen $x-y$ coordinates, the z coordinate was fixed since they simulate a roof collocation. A ROS node simulated the continuous distance estimation between each roof tag and the sensor on board of each unicycle. To ease the computational burden of the simulation, instead of the TOF technique, a simpler approach was followed: the node takes the "ground-truth" coordinates of the reference systems of the sensor and the tag, computes the true distance and then adds a gaussian noise proportional in amplitude to the measured distance. So to have less accurate measurements the farther the sensor is from the tag:

$$d_i = d_{gt} \cdot (1 + \sigma_i); \quad \sigma_i \in \mathcal{N}(0, \alpha); \quad i \in [0, m] \tag{2}$$

  with d_{gt} the ground-truth distance given by the simulation, d_i the distance of each of the m tags put on the roof, α a constant value for the standard deviation of the modelled noise.
  Once each unicycle gathers all the distance measurements from each tag then it uses them with the Odometry measurements coming from GAZEBO in a **Kalman filter** to estimate its own state.

- Distributed communication and estimation:

  As anticipated the communication network implemented is very simple, in a real case scenario it is possible to utilize UWB sensors in order to make the agents communicate in a peer-to-peer mode using the ROS software with its framework.
  As regards the situation portrayed in this project, since the communication only happens when every robot is very close to each other it was assumed to have a lossless and complete communication.
  The estimation of the target position on the plane happens with a **Least Square algorithm**, the estimation of the target height sees the implementation of a simple inverse formula knowing the distance in the plane between the robot and the target along with the vertical tilt angle of the camera.

III. SOLUTION

A.Kalman Filter Implementation

This filter was implemented in the `kalman_localization.py` script, both prediction and correction step where designed as callback functions upon reception of new data. The prediction step was invoked every time the robot receives new Odometry data from the sensors on board, the correction step was invoked every time the robots receives new data regarding its own distance from the UWB tags. I call the mean vector $\mu = [x, y, \theta]$ and assuming uncorrelation the covariance matrix

$$\sigma = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix}$$

With μ initial guess as an empty vector and σ as an identity matrix

- Prediction step

$$\mu_{k+1} = A\mu_k \tag{3}$$
$$\sigma_{k+1} = A\sigma_k A^T + GQG^T \tag{4}$$

  With equation 3 being the state prediction and equation 4 being the covariance prediction

- Correction step

$$S = (H\sigma_k H^T + R) \tag{5}$$
$$W = \sigma_k H^T S^{-1} \tag{6}$$
$$\mu_{k+1} = \mu_k + W(Z - H\mu_k) \tag{7}$$
$$\sigma_{k+1} = (I - WH)\sigma_k \tag{8}$$

  With equation 6 being the Kalman gain definition, equation 7 being the state correction and equation 8 being the covariance correction

It is intended that with k we refer to the actual value and with $k+1$ the new one. All the matrices involved will be now discussed:

- Jacobian of the dynamics with respect to the state:

$$A = \begin{bmatrix} 1 & 0 & -v \cdot sin(\theta) \cdot \Delta t \\ 0 & 1 & v \cdot cos(\theta) \cdot \Delta t \\ 0 & 0 & 1 \end{bmatrix} \tag{9}$$

  where v represents the linear velocity coming from Odometry and Δt represents the publishing period of the Odometry topic

- Kinematic model for noise redistribution:

$$G = \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \tag{10}$$

- Covariance matrix of the process, modeled as a constant value plus a gaussian term scaled with the actual value read:

$$Q = \begin{bmatrix} \nu + \nu_v & 0 \\ 0 & \nu + \nu_\omega \end{bmatrix} \tag{11}$$

- Measurement vector:

$$Z = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{bmatrix} \tag{12}$$

where z_i is the distance measured by the i-th UWB anchor

- Measurement matrix, used to map the state vector into the measurement space. It was built stacking the Jacobian of the measurement model with respect to the state. The measurement model implemented is the simple Euclidean distance:

$$d_i = \sqrt{(x - l_x)^2 + (y - l_y)^2 + (z - l_z)^2} \tag{13}$$

with l_i the tag's coordinates, and x, y, z the robots'. Since the robot always lies on the ground $z = 0$ always. Indicating with m the total number of deployed UWB tags I proceeded with the following formula:

$$H = \begin{bmatrix} H_1 \\ H_2 \\ \vdots \\ H_m \end{bmatrix} \tag{14}$$

with

$$H_i = \begin{bmatrix} \frac{x-l_x}{\sqrt{(x-l_x)^2+(y-l_y)^2+l_z^2}} & \frac{y-l_y}{\sqrt{(x-l_x)^2+(y-l_y)^2+l_z^2}} & 0 \end{bmatrix} \tag{15}$$

- Covariance matrix of the measurement process already stated in 2. It will be a diagonal matrix containing the variance of the previously stated noise, we refer now to the measurements z_i, the overall variance will be $\alpha \cdot z_i^2$. A regularization term was added to ensure the existence of S^{-1} in equation 5

$$R = \alpha \cdot I \cdot Z^2 + \epsilon \cdot I \tag{16}$$

Given the absence of prior information about the state of the robot a workaround was implemented in order to start with a decent estimate of the orientation θ.
In the code it is implemented a logic such that after a certain amount of callbacks the initial $x - y$ position is stored and then the robot is commanded to go in a straight line. After another number of callbacks the new estimate of the $x - y$ position is collected and the first θ estimate is computed as follows:

$$\theta = arctan2(\frac{y_0 - y_1}{x_0 - x_1}) \tag{17}$$

After this the node actually starts publishing the state estimate in the designated topic.

B.Least Square Algorithm Implementation

In order to estimate the target position a two step estimation was portrayed. It was computed first the plane position ($x - y$ coordinates) and lastly the z component.
To solve the first part of the problem it was necessary to have many unicycles already pointing to the target that know their own position thanks to the previously seen Kalman filter.
The first step involved considering each robot as a line in a plane exploiting its state, in other words I converted the state $[x, y, \theta]$ into a simpler and more manageable geometrical expression of a line with a slope (m) and its interception with y axis (ψ):

$$m = tan(\theta) \tag{18}$$
$$\psi = y - m \cdot x \tag{19}$$

Using the law of propagation of uncertainties:

$$\sigma_f = \sqrt{\left(\frac{\partial f}{\partial a}\right)^2 \cdot \sigma_a^2 + \left(\frac{\partial f}{\partial b}\right)^2 \cdot \sigma_b^2 + \dots} \tag{20}$$

And assuming normal distribution of x, y, θ:

$$\sigma_m = \frac{1}{cos(\theta)^2} \cdot \sigma_\theta \tag{21}$$
$$\sigma_\psi = \sqrt{\sigma_y^2 + m^2 \cdot \sigma_x^2 + x^2 \cdot \sigma_m^2} \tag{22}$$

Once all these parameters were collected for each of the n robots deployed, many attempts were executed in order to extract the target position, the most relevant:

- **Weighted pairwise interception points**: The idea was to compute every intersection point for every pair of line, compute the uncertainty and use it as a weight in a weighted average computation

$$x_{intersection} = \frac{\psi_i - \psi_j}{m_j - m_i} \quad \forall i \neq j \tag{23}$$
$$y_{intersection} = m_i \cdot x_{intersection} + \psi_i \tag{24}$$
$$\tag{25}$$

Finally the weight for the single intersection point was computed:

$$w_i = \frac{1}{\sqrt{\sigma_{x_i}^2 + \sigma_{y_i}^2}}$$

The total weight was computed and the final coordinates were drawn:

$$w_{tot} = \sum_{i=0}^{k} w_i$$

$$x_{avg} = (\sum_{i=0}^{k} x_i * w_i)/w_{tot}$$

$$y_{avg} = (\sum_{i=0}^{k} y_i * w_i)/w_{tot}$$

$$\sigma_{x_{avg}} = \sigma_{y_{avg}} = \sqrt{\frac{1}{w_{tot}}}$$

Nevertheless the good results in the end the Least Square Algorithm was chosen due to the cleanliness of the method.

- **Least Square Algorithm**: the problem is formulated as to minimize the sum of the squared distances from the estimated intersection from each line. The method takes into account the uncertainties in the slopes and y-intercepts, ensuring a weighted least squares fit.
  A single line $y = m_i \cdot x + \psi_i$ is rewritten as $m_i \cdot x - y = -\psi_i$, stacking these equations into a matrix form:

$$\begin{bmatrix} m_1 & -1 \\ m_2 & -1 \\ \dots & \dots \\ m_n & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -\psi_1 \\ -\psi_2 \\ \dots \\ -\psi_n \end{bmatrix} \tag{26}$$

$$Ax - b = 0 \tag{27}$$

  - Matrix C_{inv} definition:

$$C_{\text{inv}} = \begin{bmatrix} \frac{1}{\sigma_1^2} & 0 & \cdots & 0 \\ 0 & \frac{1}{\sigma_2^2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\sigma_n^2} \end{bmatrix} \tag{28}$$

    where $\sigma_i^2 = \sigma_{m_i}^2 + \sigma_{\psi_i}^2$ is the combined variance of the i-th line.

  - The weighted matrices are computed:

$$A_{\text{weighted}} = C_{\text{inv}} A \tag{29}$$
$$b_{\text{weighted}} = C_{\text{inv}} b \tag{30}$$

  - The least squares solution is found by solving the normal equation:

$$p = \left(A_{\text{weighted}}^T A_{\text{weighted}}\right)^{-1} A_{\text{weighted}}^T b_{\text{weighted}} \tag{31}$$

    where $p = [\hat{x}, \hat{y}]^T$.

  - The covariance matrix of the solution is given by:

$$\text{cov\_matrix} = \left(A^T \cdot C_{inv} \cdot A\right)^{-1} \tag{32}$$

  - The uncertainties in the estimated x and y positions are:

$$\sigma_x = \sqrt{\text{cov\_matrix}[0,0]} \tag{33}$$
$$\sigma_y = \sqrt{\text{cov\_matrix}[1,1]} \tag{34}$$

C.Height Estimation of the Target

Once the target's position in the plane x, y has been determined, the next step involves estimating its height z using the information from multiple robots. This estimation uses the vertical field of view (FOV) and the target heights reported by each robot.

The height estimation process is as follows:

- Having established via empirical ways that the vertical FOV is 67 degrees. At a target height value of 1, this corresponds to a 67-degree vertical angle. At a target height value of 0, the vertical angle is 0 degrees.
- For each robot, the plane distance from the robot to the target and the vertical angle are computed. Using these values, the target's height is estimated using trigonometric relationships:

$$\text{distance} = \sqrt{(x_{\text{robot}} - x_{\text{target}})^2 + (y_{\text{robot}} - y_{\text{target}})^2} \tag{35}$$

  The vertical angle δ_i for each robot is computed based on the target height reported by the robot and the vertical tilt:

$$\delta_i = \text{target\_height}_i \times \text{vertical\_FOV} \tag{36}$$

  Using the trigonometric relationship $tan(\delta) = \frac{z}{\text{distance}}$, the height of the target is estimated as:

$$\text{height}_i = \tan(\delta_i) \times \text{distance} \tag{37}$$

  The mean of all height estimates is taken as the final height estimate:

$$\text{target\_z} = \frac{1}{n} \sum_{i=1}^{n} \text{height}_i \tag{38}$$

- Uncertainty Estimation: The uncertainty in the height estimate is computed using the law of propagation of error (eq. 20) applied to equation distance37 as shown in the code.

  The combined uncertainty for the final height estimate is:

$$\sigma_{\text{target\_z}} = \frac{\sqrt{\sum_{i=1}^{n} \sigma_{\text{height}_i}^2}}{n} \tag{39}$$

  With n the total number of deployed unicycles

D.Uncertainty Considerations

Throughout the project, uncertainties play a crucial role in the unicycle state estimation and subsequent target positioning. The state estimation for the unicycle utilizes a Kalman filter which processes inputs from two primary sources: distance measurements and odometry data.

The Kalman filter combines these noisy inputs to estimate the state variables of the unicycle. The uncertainty in these state estimates is quantified using a covariance matrix, which provides a measure of the confidence in each estimated state variable.

Following the state estimation, a least squares (LSQ) algorithm is employed to determine the $x-y$ position of the target, as well as its height z. The LSQ algorithm takes into account the previously estimated states and their associated uncertainties. Using the law of propagation of uncertainties and the covariance matrix derived from the

LSQ solution, the uncertainties in the target position estimates are computed.

The input quantities (Odometry and UWB data in equation 2) are affected by Gaussian noise, but to better mimic a real implementation the noise contributions were multiplied by the actual value read. It was considered necessary to validate the uncertainties obtained from both the Kalman filter and the LSQ algorithm.
In order to do this, a type B analysis was conducted to check if the distributions and standard deviation values were close to the expected ones. The analysis revealed that the quantities derived from the Kalman filter followed a normal distribution (Example in fig. 7 and 8), indicating that the filter's assumptions and the resulting uncertainty estimates were reasonable. However, the target position estimates obtained through the LSQ algorithm did not exhibit a gaussian distribution (fig. 10, 11 and 12). This finding highlights the complexity of the noise characteristics in the target position estimates given the designed noise structure, where with high distances we obtain noisy and inconsistent data.

An example of this behavior can be shown in fig. 3 and fig. 4 where it is compared the tracking of a unicycle with more and less UWB tags deployed. In the first case, ignoring the initial inaccuracy due to the absence of a good first state guess, the predicted position tracks the ground-truth one pretty well as long as the robot doesn't get too far from the tags, when this happens the data coming from the UWB anchors presents noise proportional to the distance and the estimation process presents a more inconsistent behavior. In the latter due to the addition of two more tags on the right the estimation process is cleaner and more robust, it is also possible to notice how even in the presence of an outlier the filter manages to contain the deviation and comes back to a good estimate in two algorithm steps.

The type B uncertainty evaluation were portrayed simulating the deployment of 5 unicycles and 6 UWB tags, a single target of coordinates $1,2$ $[m]$ and height 0.85 $[m]$. All robots managed to reach the same estimate demonstrating the efficacy of the distributed estimation algorithm. In fig. 6 is shown the overall plot of each robot's own estimation of target's z coordinate, apart from some random spike they all overlap meaning that they all reach the same results in a distributed manner.

It's possible to see how not all robots reach the target at the same time, hence the number of agents contributing to the distributed estimation changes with time. One of the first to arrive on the spot is the robot n°4, for this reason it was chosen to plot its estimates. As previously stated it's possible to see the goodness of the Kalman filter estimation judging by the distributions in fig. 7, 8 and 9, the mean error is practically non existent and the average expected standard deviations for the state components are:

$$\sigma_x = 0.088\ [m] \tag{40}$$
$$\sigma_y = 0.088\ [m] \tag{41}$$
$$\sigma_\theta = 0.839\ [rad] \tag{42}$$

The actual average standard deviations are coming from type B uncertainty analysis:

$$\sigma_x = 0.043\ [m] \tag{43}$$
$$\sigma_y = 0.043\ [m] \tag{44}$$
$$\sigma_\theta = 0.774\ [rad] \tag{45}$$

As regards its estimation of the target the expected standard deviations are in the order of 50 centimeters for x and y. For this reason only the latter kind will be portrayed with respect to the distributions in fig. 10, 11 and 12.

$$\sigma_{x_t} = 0.033\ [m] \tag{46}$$
$$\sigma_{y_t} = 0.024\ [m] \tag{47}$$
$$\sigma_{z_t} = 0.017\ [m] \tag{48}$$

It is finally worth noticing the peculiar distribution of these plots, it is explainable by the fact that the estimation is dynamic and not 100% precise, each peak corresponds to the average estimation error when a different number of robots participates. As an example in fig. 11 we can see 3 main peaks corresponding to the error when the robot estimates the target with other 2 robots and respectively when the 4th and the 5th one join the distributed estimation.

IV. PRACTICAL IMPLEMENTATION DETAILS

The development of the project saw many obstacles regarding its implementation, in particular a lot of time and effort was spent in order to create a project that would be the most modular possible. An example is the `URDF` section and the logics in the launch files to guarantee dynamical simulations with the minimum effort possible regarding its setup.

A. Overall Project Structure

The whole project has been developed and constantly submitted to a Github repository referenced in [2]. Several `README` files and a lot of commentary inside the python scripts have been prepared to guide future users into the understanding of its structure under a more technical point of view. The vast majority of the logic is implemented in the `scripts` folder, where the python executables are stored.

Due to its modularity the hierarchy of the communicating topics could be confusing, for this reason in fig. 1 and fig. 2 are presented the graphs of the system in two distinct phases:

- Movement towards target in fig. 1:
  It is possible to notice how the ground truth topic feeds the UWB tag simulation node, the localization nodes if fed by the Odometry node coming from gazebo and by the uwb data node. At this step the motion planner still waits for the start signal from the `init_move` topic, it will then publish the unicycle control input to the `cmd_vel` topic that directly forwards everything back to the gazebo simulation.

- Target estimation in fig. 2:
  In this situation the unicycle is already on target, the motion planner node does not need any input and also publishes the read target height. The target estimator node now publishes and reads data from the common topic `processing_data` that in this project simulates the communication between different robots.

As regards the image processing node a few notes need to be added: the final script doesn't show a lot of lines of code that were added to print the middle results of each filter for debug reasons. It was decided to delete them in the final version since the file would have been non readable otherwise.

Every other file has been fully commented in every part, a last comment on the state machine implemented in the motion planner script need to be considered: The motion planner handles the motion of the unicycle switching between 4 states.

1. *INIT*: In this first phase the robots goes in a straight line in order to start with a better state estimate as previously described in the previous paragraph
2. *SEARCH*: In this state the robots turns on itself waiting for the image processing node to detect the target from its camera.
3. *APPROACH*: In this state the node uses the target coordinates in its camera reference system to guide the robot towards it. A simple *PID* controller has been implemented to steer the unicycle, the forward velocity v was kept constant

$$\omega = k_p \cdot e + k_d \cdot \frac{de}{dt} + k_i \cdot \int e dt \tag{49}$$

   The error e was set to be the x coordinate of the target in the camera point of view since it is desired that the unicycles directly point at the target while going at a constant speed. In addition to this in this project the k_i constant has been kept to 0 since there was no big offset error to correct.
4. *TARGET_REACHED*: In this final state the motion planner stops the unicycle and sends the necessary messages to start the target estimator node.

V. CONCLUSIONS AND FUTURE WORKS

A. Communication and real-world improvements

To address some of the weaknesses exposed at the beginning a few counter-measures have been proposed:

- *More robust communication protocol*: To enhance the UWB communication with protocols that handle interferences and provide reliable data transmission an idea would be the transition from ROS to ROS2 that with its communication middleware *"DDS: Data Distribution Server"* offer improved performances and reliability. In addition to that ROS2 offers *QoS: Quality of Service* settings out of the box for an easier communication configuration.
- *Redundancy and error correction*: some custom nodes could be created to handle retransmission of lost messages, maybe implementing some protocol like *"FEC: Forward Error Correction"* that by adding redundancy to the transmitted data allows the receiver to correct errors without needing a retransmission or *"ARQ: Automatic Repeat reQuest"* that implements an acknowledgment infrastructure between communicating nodes.
- *Adaptive networking*: a dynamic approach to managing network resources that adjusts to changing network conditions to maintain optimal performance. It involves monitoring the network's state and making adjustments to parameters such as bandwidth allocation, routing paths . . . It can all be easily implemented in ROS using the `dynamic_reconfigure` python package
- *Signal strength monitoring*: some custom node monitoring the signal strength and quality of UWB connections could work in parallel to the previous adaptive networking logic

B. Additional theoretical implementations

For future theoretical implementations, one promising approach could be the integration of a linear consensus algorithm to refine the final target position estimate. One obstacle that should be dealt with however is the synchronization of this algorithm. Ignoring this would lead to considerable delays in its convergence that could make it not worth using in a real-case scenario.

To ensure synchronization in the ROS environment, a possible solution would be to implement a synchronized communication protocol, such as the ROS Time Synchronizer, which aligns the timestamps of the messages exchanged between nodes. In addition to that, by incorporating clock synchronization techniques like the Network Time Protocol (NTP) or the Precision Time Protocol (PTP), would further guarantee that all unicycles operate on a consistent time frame, enhancing the reliability of the consensus algorithm in real-time applications.

C. Conclusion

The main aim of this project was to build a consistent framework for future projects in this subject. Particular attention was given to ensuring that everything was as modular and understandable as possible for future users. Despite the limited theoretical implementations, this project provides a solid foundation for any kind of future development.

The simulations conducted in this project demonstrate that good results were achieved, including accurate target estimation and a reliable localization filter utilizing UWB tags. Another strong point is the close alignment between simulation and real-world implementation, as both environments share the same software infrastructure. This consistency ensures that the transition from simulated scenarios to practical applications will be seamless and effective.

VI. PLOTS

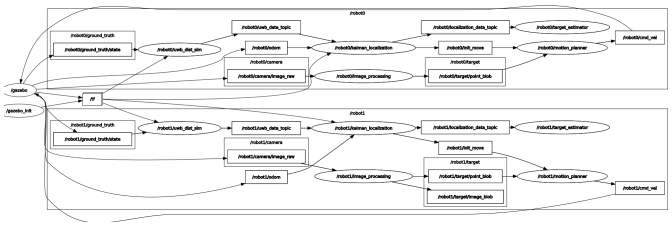

Figure 1: Graph of topic hierarchy before target estimation phase

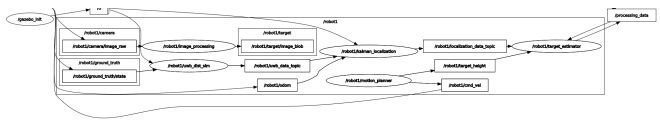

Figure 2: Graph of topic hierarchy during target estimation phase

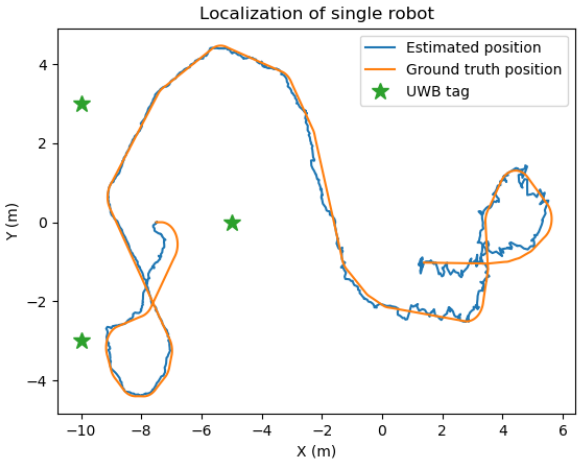

Figure 3: Ground-truth and estimated position, starting position $(-7.5, 0)$, 3 tags deployed

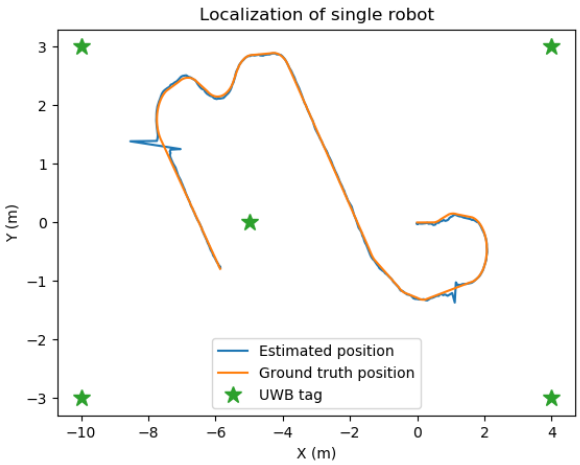

Figure 4: Ground-truth and estimated position, starting position $(0, 0)$, 5 tags deployed

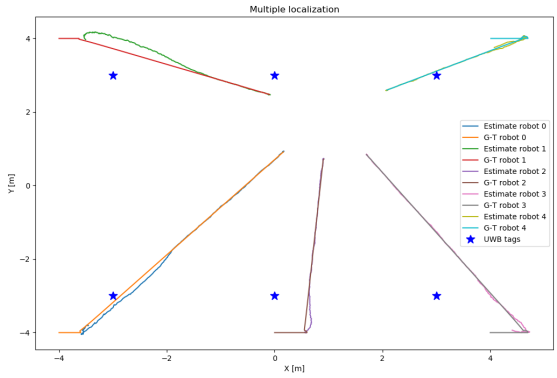

Figure 5: Simulation of 5 unicycles, 6 UWB tags and 1 target of coordinates $1, 2$ $[m]$

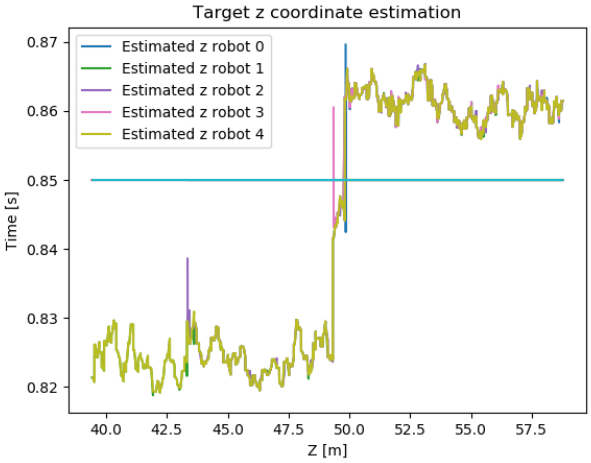

Figure 6: Overall plot of every estimate of target's z coordinate

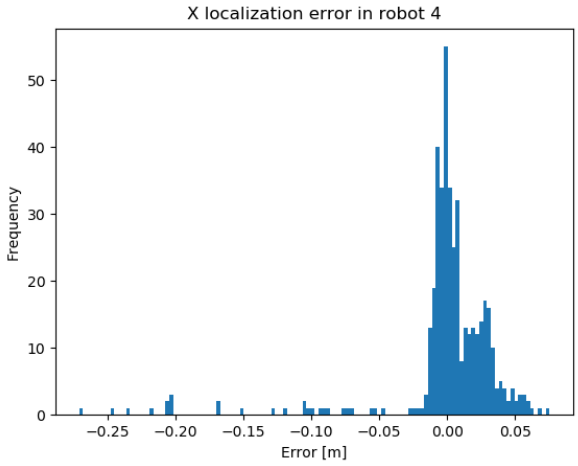

Figure 7: Estimate of robot 4 x coordinate

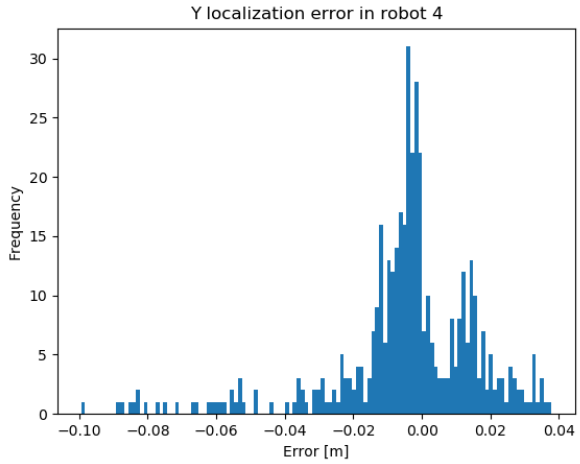

Figure 8: Estimate of robot 4 y coordinate

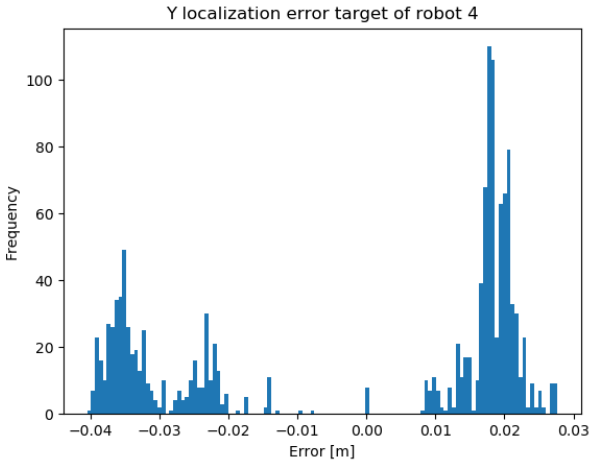

Figure 11: Estimate of target's y coordinate by robot 4

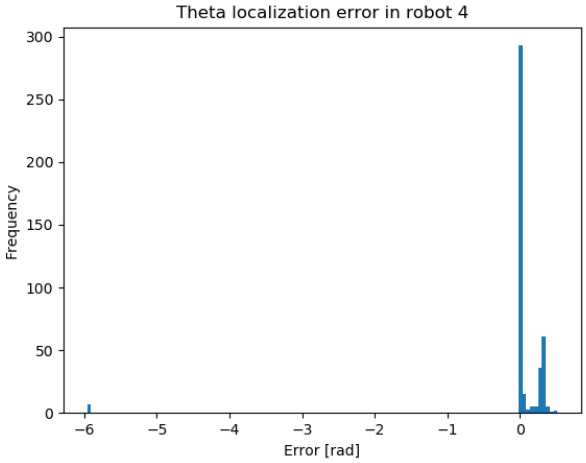

Figure 9: Estimate of robot 4 orientation θ, the spike is close to -2π

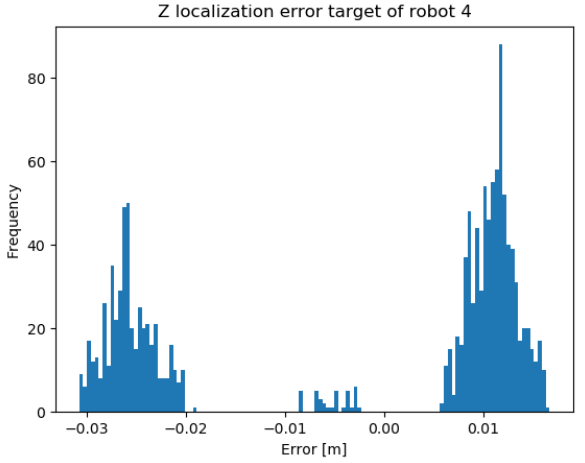

Figure 12: Estimate of target's z coordinate by robot 4

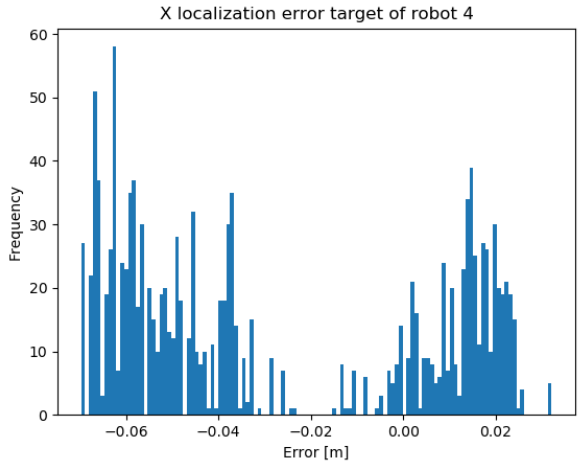

Figure 10: Estimate of target's x coordinate by robot 4

REFERENCES

[1] Daniele Fontanelli et al. "An Uncertainty-Driven and Observability-Based State Estimator for Nonholonomic Robots". In: *IEEE Transactions on Instrumentation and Measurement* 70 (2021), pp. 1–12. DOI: 10.1109/TIM.2021.3053066.

[2] Marco Sterlini. *Distributed project repository*. 2024. URL: https://github.com/Marcosterlo/Distributed_project.git.

The author have the following address: Italy,
marco.sterlini@studenti.unitn.it
This report is the final document for the course of "Distributed Estimation for Robots and Vehicles".