Multimodal Navigation for Autonomous Service Robots

Rui Bettencourt
rui.bettencourt@tecnico.ulisboa.pt
Instituto Superior Técnico, Universidade de Lisboa, Portugal
November 2019

***Abstract*—Service robots for indoor environments is a very promising field, since it may solve many of the problems originated by the aging demographics in developed countries. These robots can assist people with mental or physical limitations or help anyone in their homes or workspaces, allowing them to use their time more productively. Thus, service robots need to be able to navigate in such environments, which might include narrow passages, and at the same time avoid, seamlessly, any static or moving obstacles. Moreover, the robot should be able to localize itself at all times to correctly perform its tasks. Since it is not possible to guarantee this, a safety measure to recover from the kidnapped robot problem should be implemented. In this thesis, a method is proposed to solve this challenge using state of the art algorithms with additional implemented components to create a fully operational navigation stack. To enable non-static goals, we introduced a dynamic goal component that follows a moving goal to its exact pose or to try and keep a certain distance from it. This dynamic goal can be an object, a person or any element that can be tracked over time and produce a pose estimation to be followed. The proposed method was implemented inside the framework SocRob@Home with the MBot robot. Using the robot in experiments at the ISRoboNet@Home tesbed and in the competition ERL Smart Cities this thesis presents how the developed navigation stack behaves correctly.**

***Index Terms*—Navigation, Localization, Service Robots, Indoor spaces, Guiding, People Following, Waypoint Navigation, Dynamic Goal**

I. Introduction

Service robots have been studied and developed for over thirty years [1] and one of the most important components of every robot with mobility is its navigation. All mobile robots require a good navigation stack to have a correct behaviour and it can operate in different modes with different navigation objectives (Multimodal Navigation). The case study for this thesis is ISR's MBot[1] used by the team SocRob@Home [2] where a robust, efficient and intelligent navigation stack is a crucial component so that the robot is able to perform several indoor tasks with the goal of helping people in their daily lives. The results of this project are of great importance in competitions such as the European Robotics League (ERL) Consumer Service and Smart Cities Robot Challenges, or RoboCup@Home [3] where several students can implement their research in specific challenges. Most of the navigation modes described in this thesis were implemented in the robot during ERL Smart Cities competition.

[1]from the FP7 MOnarCH robot

The main objective of this work is to implement a working multimodal navigation with the following modes: 1) Waypoint navigation, where the robot receives a static goal and has to plan the clearest and shortest path to this goal in a known map and navigate to it while avoiding obstacles; 2) Dynamic goal navigation, where a goal that can change pose is sent to the navigation stack instead of a static one. This goal can be a simple pose in the map that can change its position and/or orientation in each time instant, e.g. the position of a tracked object or the position of a tracked person to follow; 3) Guiding navigation, where there is a static or dynamic goal that the robot must reach whilst making sure that a certain tracked person is following the robot. These modes are implemented with existing state of the art components and with new implementations in order to accomplish all the different modes of navigation. The main contribution of this work is a complete navigation stack that includes the dynamic goal package to follow a moving target pose. The rest of the document is structured as follows: Section II presents the related work, Section III explains the theoretical background of the used methods, Section IV presents the multimodal navigation method, Section V shows the obtained results and discusses them, and Section VI presents the conclusions and future work.

II. Related Work

The navigation loop diagram used can be seen in Fig. 1.

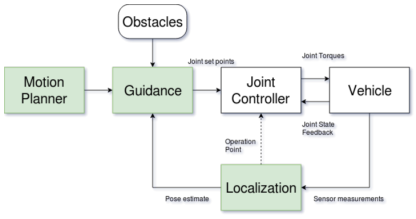


Fig. 1. Diagram of the navigation loop [reprinted from [4]].

The *Localization* block supplies the *Guidance* block the necessary pose estimate so that it can create a trajectory for the *Joint Controller*. *Localization* also supplies the operation point that *Joint Controller* works on.

Localization has two components, a global localization and a local position tracking. The global localization is performed when the robot does not know its initial pose and needs to

determine it whereas local position tracking is the component that updates the localization as the robot moves in the map over time. Besides these two components, the kidnapped robot problem also exits as a version of the global localization with an additional difficulty of having to determine when the robot is or is not well localized and recover the correct localization if it is wrongly localized [5].

To study an uncertain environment, "the most general algorithm for calculating beliefs is given by the Bayes Filter" [5].

The most "straightforward application of Bayes filter to the localization problem is called Markov localization" [5]. This method is an algorithm that keeps a probabilistic distribution for the robot pose instead of a single hypothesis [6].

A well studied case of a Bayes filter is Kalman Filter [7], useful for linear systems. It uses the mean μ_t and the covariance Σ_t to represent the belief and is not applicable for discrete or hybrid states, working only for continuous states [5]. To extend the Kalman filter to be used in nonlinear problems, the Extended Kalman Filter (EKF) was introduced [5]. EKF used for localization is a special case of the Markov localization where the measurement model has been linearized with Gaussian noise [5].

As it is presented in [8], another method and arguably the most popular, Monte Carlo Localization (MCL) is a sample-based representation and it is able to represent multi-modal distributions, it is able to be used in both global and local localization as opposed to EKF and it is more efficient than Markov localization since its computational cost does not depend on the size of the map, but on the number of particles. Although in its original version it could not recover from the kidnapped robot problem, a new version was introduced that can recover from this problem by adding random particles to the particle set when the localization accuracy is low. This new version was named Augmented MCL. To optimize the method, the sampling can be done in an adaptive manner [9] that changes the number of used particles depending on how spread out the particles are.

The motion planner is the part of the navigation that given an origin and target poses will give a path of a continuous sequence of poses that connects both taking into account the global costmap. In Fig. 1 the integration in the navigation loop of this component is shown, where it supplies a path to a certain goal to the *Guidance* block to follow.

Several path planning approaches are used for motion planning such as: 1) Roadmaps; 2) Cell Decomposition; 3) Sampling-Based Planning; 4) Potential Field; 5) Bug Algorithms [10] [11]. The two main algorithms used as shortest path planners are Dijkstra's algorithm and A* algorithm [12]. Analysing both algorithms like it was done in [13], Dijkstra's algorithm combined with gradient descent method allows the smoothest path. A* is the most efficient method but might not result in the optimal path in terms of smoothness, since this method does not look into every nearby map cell that is used in gradient descent method to get the optimal smoothest path.

In Fig. 1, the *Guidance* block takes the path from the *Motion Planner*, the pose estimate from *Localization* and the detected *obstacles* and creates a trajectory (e.g., wheel velocities) for the *Joint Controller* to apply. Like it is mentioned in [13], some state of the art guidance algorithms are Dynamic Window Approach (DWA), Elastic Band (EBAND) and Timed Elastic Band (TEB). As it is shown in [14], DWA determines a velocity command from a determined set of velocities that can be reached by the robot according to its dynamic and kinematic constraints.

For people tracking, in [15], the authors focus on the tracking of a single operator and on the re-identification of this person. This is done so that people following can always track the same person, even in cluttered spaces and even after loosing the person for a while. In order to make robot navigation more human aware, the article [16] proposes the merge of laser scans and RGB-D cameras in order to detect humans, fusing the several detectors with a Kalman filter.

In terms of people following, a people follower is presented in [17], succeeding in following using distances defined by the notion of Proxemics [18], but fails to back away to maintain its distance if the person gets closer to the robot than it should.

The method proposed in [19] can track and follow a person with a navigation stack built specifically using a PID to calculate the angular velocity and one PID to calculate the linear velocity. The method is based on the comparison between two vectors, one from the robot center to the tracked person and one from the robot to the desired goal position from the robot to the person. The angular and linear velocity commands have the goal of equalizing the length of the vectors and bring the angle between them to zero.

Another possible way to follow a person is to track the person and aim to follow its tracked path, maintaining a distance to the person as it is done in [20].

III. Background

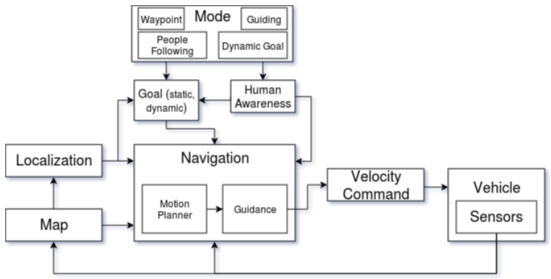


Fig. 2. Diagram of the overall behaviour of the several components of the proposed navigation.

In Fig. 2 the different components required for navigation are represented. It needs a localization component, a map and sensor reading to navigate correctly. The navigation needs two main components, the motion planner and the guidance component to reach a certain navigation goal with the shortest distance and without hitting any obstacle. The mode of navigation is chosen from the four modes presented in the diagram

that will automatically define which type of goal is required and if Human Awareness is needed.

A. Covariance and Variance

If a measurement unit for how much a single variable varies (or measurement uncertainty) is needed, variance can be used [5]. Another possible measurement unit is the standard deviation, that is the square root of the variance. A variance is defined as in Equation (1), where $E[x]$ is the $expectation$ of a discrete-space variable x expressed as in Equation (2), in discrete time. For every discrete-space variable x, there is a specific event x_1 that x might result on, that has a probability $p(x_1)$ of becoming true.

$$Var(x) = \sigma(x,x) = E\Big[\big(x - E[x]\big)^2\Big] = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2 \tag{1}$$

$$E[x] = \sum_{x_1} x_1 \; p(x_1) \tag{2}$$

To measure how much two variables influence changes in one another, covariance is used. The covariance of two variables x and y is given by

$$\sigma(x,y) = E\Big[\big(x - E[x]\big)\big(y - E[y]\big)\Big]. \tag{3}$$

To represent all the variances and covariances, a covariance matrix is built. This matrix is symmetric and positive-semidefinite, with the variances of the variables in the diagonal and the covariances between the different variables in the other positions of the matrix. A covariance matrix for two variables x and y can be seen in Equation (4).

$$\Sigma = \begin{bmatrix} \sigma(x,x) & \sigma(x,y) \\ \sigma(y,x) & \sigma(y,y) \end{bmatrix} \tag{4}$$

B. Adaptive MCL

To represent an approximate calculation of the robot's state the concept of *belief* is introduced. Belief is used to have a probabilistic representation of the environment's state taking into account the sensors measurements and the control commands up until the present time, since the algorithms can not know directly the true state of the robot.

To calculate beliefs a very general algorithm is given by the Bayes filter, that calculates the belief distribution from the measurements and the control data. Monte Carlo Localization (MCL) is a localization algorithm that represents the belief $bel(x_t)$ of a robot's location by particles. It works efficiently both in the global and local localization but it cannot recover from the kidnapped robot problem or from wrong localizations.

To solve this limitation of the algorithm, another version of the algorithm was presented, Augmented MCL. This new version proposes a simple heuristic to solve the problem that can be summed up by the addition of random particles to the set when the robot perceives it is localized in a wrong location by checking the measurement's likelihood. The added random particles are proportional to the estimation of the location accuracy. This estimation of the robot's localization accuracy can be obtained from the probability of sensor measurements

$$p(z_t | z_{t-1}, u_t, m) \tag{5}$$

in relation to the average measurement probability. Since in particle filters the importance weight can be considered a stochastic estimate of Equation (5), the mean

$$\frac{1}{N}\sum_{n=1}^{N} w_t^{[n]} \tag{6}$$

can be used as the estimation of the measurement likelihood. Since this average can be always high due to, for example, noisy sensors and that there is no reference value to judge if this average is good or not, it is better to have a long term average estimation of the accuracy of the location and a short term one and compare both to determine the deterioration of localization accuracy.

This solution to the kidnapped robot problem implemented in the Augmented MCL algorithm differ from MCL by declaring 2 variables: w_{slow} and w_{fast}. These variables are the average of the long-term and short-term averages, respectively. This means that the w_{slow} value takes into account past iterations with a bigger importance than w_{fast}, that represents mostly the few last iterations, hence the name short-term average.

The importance that each of these averages give to each new iteration is defined by the two constants α_{slow} and α_{fast}, that influence their respective average. For the algorithm to work as it is designed to, the following condition needs to be true: $0 \leq \alpha_{slow} \ll \alpha_{fast}$. The reason why this relation has to be true is to accomplish the long-term and short-term component. By being much greater than α_{slow}, the constant α_{fast} impacts much more the value of w_{fast} in each iteration than α_{slow}, making it the intended short-term average.

The empirical measurement likelihood that was presented in Equation (6) is calculated and used for the update of the variables w_{slow} and w_{fast}.

The resampling loop is also different, where a probabilistic condition is added to add random particles to χ_t. The way it works is that for each N iteration of the resampling loop either a completely random sample is added to χ_t or a particle from $\bar{\chi}_t$ is chosen. This randomization or not of a particle is decided by a random condition with the probability

$$max\Big(1.0 - \frac{w_{fast}}{w_{slow}} \;,\; 0\Big). \tag{7}$$

What this probability means is that if the short-term average is much greater or equal to the long-term average, there is no need to add random particles since the robot is likely well localized. If the short-term average is higher than the long-term one, then the relation between them is $0.0 < w_{fast}/w_{slow} < 1.0$, and random particles will be added proportionally to this relation.

Another possible optimisation is to vary the number of particles according to what is needed at each iteration of the algorithm. At each iteration, the Kullback–Leibler divergence (KLD) approach samples particles until their number is high enough to guarantee with probability $1 - \delta$ that the Kullback–Leibler distance between the maximum likelihood estimate and the belief estimate does not exceed a predefined limit ϵ. The result of this approach is that there is a smaller number of particles if most samples are focused in a small area and more particles if the samples are more distributed along the state space.

The Adaptive MCL will then stop when the number of samples n is no longer part of the interval defined by the expression

$$min_p < n < \frac{1}{2\epsilon}\chi^2_{k-1,1-\delta} \tag{8}$$

where k is the number of bins (sections of the map that are different from each other, similar to map cells), min_p in the minimum number of particles and the values of δ and ϵ are fixed parameters.

C. Dynamic Window Approach

The Dynamic Window Approach (DWA) is a guidance algorithm proposed in [14]. This method obtains its trajectory by computations made in the space of velocities. Considering the search space with all possible velocities V_s, the search space used in DWA after the restrictions imposed above can be obtained by

$$V_r = V_s \cap V_a \cap V_d \tag{9}$$

where V_a is the set of velocity vectors (v,w) that do not hit any obstacle and V_d is the set of velocities achievable for the robot taking into account its dynamic constraints.

After obtaining the final search space obtained by Equation (9), the optimal velocity is obtained from an objective function that considers the heading of the robot towards the goal, the distance to the closest obstacle and the linear component of the robot's velocity.

D. Dijkstra's Algorithm

To solve the shortest-path problem on a weighted graph, Dijktra's algorithm can be used, as it is shown in [21]. A weighted graph means that the graph has several nodes connected to each other, and the connections have an associated weight. This weight is the cost of the transition from a node to another.

The current node is set as the start node in the beginning, it proceeds to check all unvisited neighbour nodes calculating the Δ (weight of nodes transition) to the current node. This new Δ is compared to the current Δ of the node and the smallest value is kept. The Δ is calculated as the Δ of the current node plus the weight to the unvisited node. When all neighbour nodes are checked, mark the current node as visited and select the node that has the smallest Δ and is unvisited. The algorithm ends when the goal node has been visited or when all remaining unvisited nodes have $\Delta = \infty$, which means that the remaining unvisited nodes cannot be reached.

When the algorithm is finished, the path from the origin to the goal can be obtained tracing back from the goal node by checking the parent node of each node in the path until it reaches the origin. This algorithm's result is an optimal shortest path solution for the selected graph, origin node and target node, and it will find a solution if the origin and target node belong to the same graph tree.

IV. Multimodal Navigation

Robotic Operating System (ROS) is a framework that facilitates the integration of robot software. Its libraries, tools and conventions simplify the task of working with robots.

Currently, the most used navigation package in use on ROS framework is *navigation*. This package includes several others such as: 1) *amcl*; 2) *move_base*; 3) *map_server*; 4) *global_planner*; 5) *costmap_2d*; 6) *base_local_planner*.

A. Coordinate Frames

A robotic system has many components that move in relation to each other and it is important to know the position of each component in relation to the others, specially because most measurements are taken in different frames and need to be transformed to a same frame to allow a combination of the measurements. To solve this problem, the framework ROS has a tf library [22]. The idea of this library is to connect every frame of the system in a tree graph structure such that each frame is a node and the transform between two frames is represented as edges of the graph. . This library allows to get for a certain component the transform from the coordinate frame of another component if those two components are in the same graph.

To perform the frame transform operation between two tfs, a reference frame and a target frame are required. The transformation returns the translation and rotation of the target frame in relation to the reference frame. In the used implementation the base tf tree used can be seen in Fig. 3.

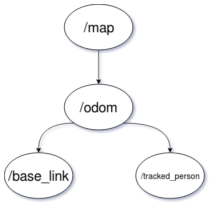


Fig. 3. Base tf tree of the implemented navigation method.

B. Localization

The objective of a robot's localization is to be able to localize itself properly and maintain the location when it moves. An indoor environment can be a domestic environment or a professional workspace and in both the robot requires localization.

The localization algorithm must also be able to handle the kidnapped robot problem whether it is indeed moved by someone or even if it initially chooses a wrong location.

In either case, it needs to recover to its correct localization to perform the tasks. Another factor is the resources this algorithm will use.

For these reasons, the algorithm that is used for localization is Adaptive MCL for its global and local localization accuracy, its efficient amount of spent computational resources and the ability to recover from wrong localizations.

The algorithm uses the map generated by FASTSlam v2 [23](implemented by gmapping), the laser scans and the transform tree to output a pose estimation. The used amcl parameters were: 1) Particles from a minimum of 100 to a maximum of 20.000; 2) Minimum translational distance moved before updating the filter: 0.05 m; 3) Minimum rotational movement before updating the filter: 0.01 rad/s; 4) Number of filter updates before resampling: 2; 5) Value of α_{slow}: 0.001; 6) Value of α_{fast}: 0.1.

The values of α_{slow} and α_{fast} are very important since they influence how fast the algorithm recovers from a wrong location but also how often it realizes that its location is wrong. This can cause the robot to always question its location and change it often, preventing a smooth and continuous movement. The chosen values were obtained and tested experimentally so that the robot was able to recover, even if taking more time, and simultaneously ensuring that the movement from navigation could be more fluid.

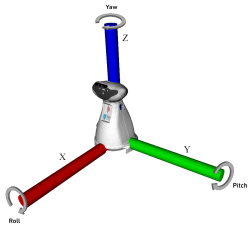


Fig. 4. Robot's reference axis

The robot has 3 degrees of freedom, translation on x, y and rotation expressed as an Euler angle yaw, as seen in Fig. 4. The covariance matrix of the translation variables is given by the covariance matrix

$$\Sigma_2 = \begin{bmatrix} \sigma(x,x) & \sigma(x,y) \\ \sigma(y,x) & \sigma(y,y) \end{bmatrix} \tag{10}$$

and the variance of yaw is given by $\sigma(yaw, yaw)$.

C. Navigation

The navigation stack are all the components necessary to choose the path towards the desired goal and decide the velocity commands to send the motors to follow this plan. The architecture scheme of the proposed navigation stack can be seen in Fig. 5.

1) Target Goal: In order to achieve our objective, the navigation should be able to receive not only static goal poses but also dynamic ones. By using the tf library, it is possible to always have the pose of a certain tf even if it moves between two time instants. If a dynamic object or person that

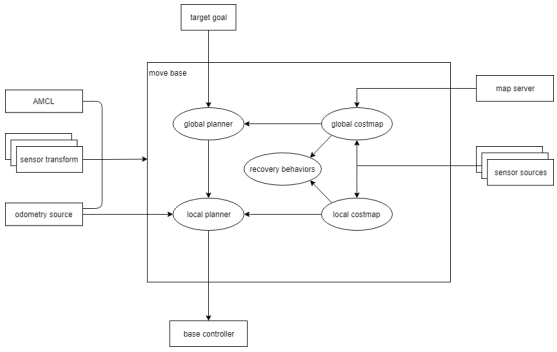


Fig. 5. Navigation's architecture Diagram [reprinted from [24]]

should be followed is represented in a frame, by calculating the transform to a reference frame like /map the pose of this dynamic target is known. If these two frames exist and are known, the transform between them results in the translation and rotation between frames.

By implementing the dynamic goal using coordinate frames, the default reference tf (origin_tf) will be */map* but the target frame (dyn_goal) has no default since it is not known which frame the algorithm should follow. This means that this parameter is required as an input for the algorithm to work.

TABLE I
DYNAMIC GOAL MESSAGE TYPE

type	denomination	optional/needed
bool	activated	needed
string	dyn_goal_tf	needed
string	origin_tf	optional (default is /map)
float64	dist	optional (default is 1.2 m)

A new message type is introduced in table I that is used as an option to the static goal in move_base package. With this new message defined, the two following goals are accepted:

1) $/move\_base\_simple/goal$ - static goal topic
2) $/move\_base\_simple/dyn\_goal$ - dynamic goal topic

The static goal receives a pose and the dynamic one receives the message type described in table I. These two options are mutually exclusive which means that they cannot be used at the same time. As seen in table I, the dynamic goal has a distance parameter to define how far the robot should try to be from the goal. This value could be 0 if the robot should reach the target frame or a positive distance if the robot should keep a certain distance from the target.

If $r > 0$ (r is the radius of the circle and the distance to the dynamic goal) the robot does not have a defined goal pose but an infinite set of poses that can be set as the goal. To choose a pose from this infinite set an iterative process is introduced to calculate the best pose. This iterative process that is followed to determine this *best pose* is:

1) A circle of radius r is created around the target with a certain number of points in the circle $n_{pc} = r \cdot 50$.
2) All the points are ordered by the distance to the robot.

3) One by one, the points are evaluated if they are a free cell in the costmap or not. If they are not, the point is discarded and the next point in the ordered list is evaluated.
4) If the point can be set as a goal, the algorithm checks if there is any obstacle between the robot and the target. If there is, the point is discarded and the next point is evaluated.
5) The closest point to the robot that satisfies these conditions is set as the *best pose*.
6) The orientation the robot should have is set so that $base\_link$ is oriented towards the target pose.
7) The pose with the chosen position and orientation is chosen as the navigation goal for this dyn_goal at the current iteration.

This process can be seen in Fig. 6 and is performed every time the robot perceives that the target frame has moved at least a certain amount defined in a threshold or until the dynamic goal mode is deactivated.

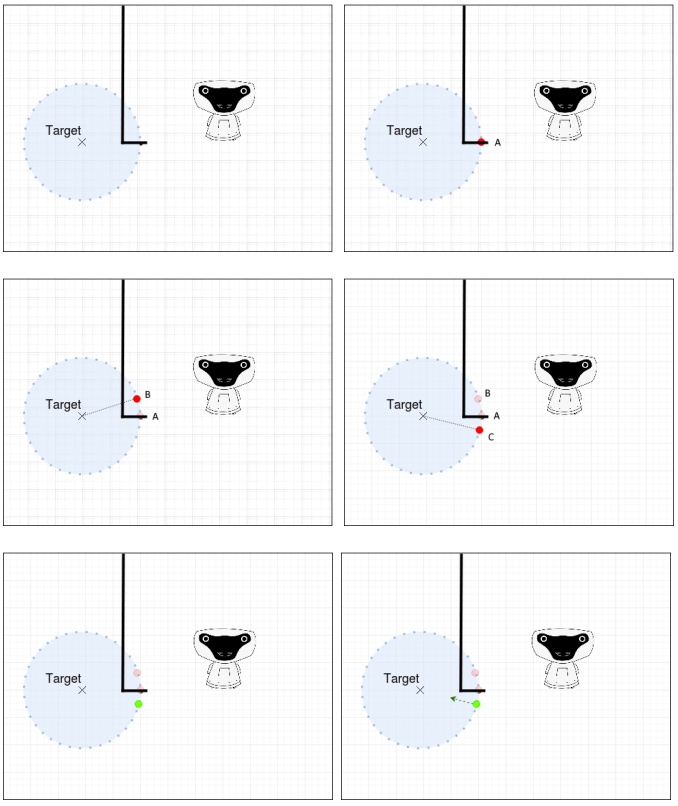


Fig. 6. Selection of the optimal position

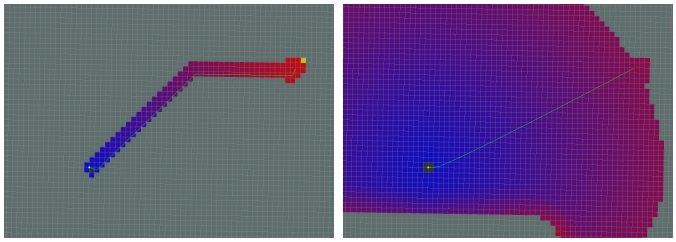
In the proposed iterative process to choose the goal pose of the robot, in step 4, the way the method checks if there is a clear path towards the target is by making a straight line from the point being analysed and the desired target. This line is made of a finite sample set of points composed of a number of points that vary according to the chosen r. This number of points is given by

$$n_{pl} = r \cdot 20. \tag{11}$$

where the points are equally distanced from each other by

$$\delta = \frac{x_{target} - x_{origin}}{n_{pl}}. \tag{12}$$

2) Global Planner: The chosen global planner from the motion planners presented is a cell decomposition algorithm with Dijkstra's algorithm as the path planning algorithm. The reason why Dijkstra's algorithm was chosen and not A* was to have the smoothest and optimal path over the best efficiency, as can be seen in Fig. 7.



(a) A* **(b)** Dijkstra's algorithm

Fig. 7. Comparison between the two mentioned shortest path planners [25].

3) Local Planner: The local planner chosen was DWA since it can correctly choose a velocity command to send to the base controller, following a global path and is able to avoid dynamic obstacles.

The main configurations used for this component are related to the dynamical constraints of the robot, setting the limits of the robot's speed and acceleration for translation and rotation. Instead of setting these values only once, it might be necessary to change the top speed of the robot depending on which action the robot should perform. For this, several configurations were made to easily change the value of these Adaptive MCL parameters in runtime. These configurations are important to produce different types of movements for the navigation modes in order to improve the robot behaviour according to the selected mode.

D. People Following

People following is an example of a dynamic goal, since to follow a person the navigation stack needs to constantly track this person, determine its position and navigate towards it or back away from it, according to the defined parameters. The detecting and tracking component used is originated from previous members of the team SocRob@Home. From this component, the tf frame *tracked_person* indicates the pose of the tracked person to be followed by transforming to the */map* frame.

Taking into consideration the Proxemics theory [18] the distance of 1.2 meters was used in the tests and experiments, since it is the limit between the personal and the social space.

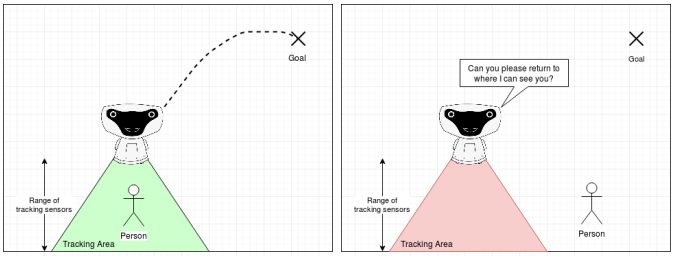
This is everything that needs to be done to run the people following mode if the dynamic goal component is integrated in the system. The message sent to the topic $/move\_base\_simple/dyn\_goal$ can then be, for example, as simple as in Listing 1 and to deactivate it, a message with *activated=False* needs to be sent.

```
dyn_goal_msg(
    activated = True,
    dyn_goal_tf = "tracked_person",
    origin_tf = "map"
    dist = 1.2)
```

Listing 1. Message to activate people following

E. Guiding

Guiding mode, like the people following mode, requires the component of human perception. To perform this mode, a guiding mode was designed so that the robot first verifies if there is someone being tracked and awaits a person. When the person starts being tracked, the robot will aim to keep tracking them while moving toward a predefined goal.



(a) Guiding a person **(b)** Waiting for the tracked person

Fig. 8. Guiding diagram

To start the movement, the robot sends a goal to move_base with the configuration *only_back* (only allows the robot to move backwards) so that the robot moves towards the goal while maintaining the target person in front of its camera, necessary for the tracking. The robot then calculates, at each time instant, the distance to the tracked person and if it lost them. In these cases, it will stop the movement and say something to attempt to bring back the person in front of it or simply wait for the person to catch up. This mode keeps running until the goal is reached.

V. Results and Discussion

The results obtained and discussed in this Chapter demonstrate the navigation stack capabilities and robustness as it undergoes multiple tests. The experiments are divided into four main components: 1) The localization, where this component is tested to see if it can maintain the location and if it can recover in case of a kidnapped robot situation; 2) The waypoint navigation, that should correctly plan the shortest path to a goal and travel to it while avoiding obstacles and maintaining its localization; 3) The people following mode, that should aim to keep a certain distance to the tracked person; 4) The guiding mode that should successfully navigate to a waypoint while making sure that the tracked person is following it.

These results obtained were recorded from experiments performed either in the ISRoboNet@Home testbed or in the ERL Smart Cities competition challenge, where the navigation stack was used. The robot used was MBot robot from MOnarCH project and produced by IDMind, with 4 Mecanum omnidirectional wheels allowing it to move in any direction.

As for sensors, besides the camera and the microphone, the robot has two Hokuyo URG-04LX-UG01 lasers, one at the front and one at the back, that allow it to have almost 360 degrees of laser sweep coverage, if not for some blind spots next to the sides of the robot.

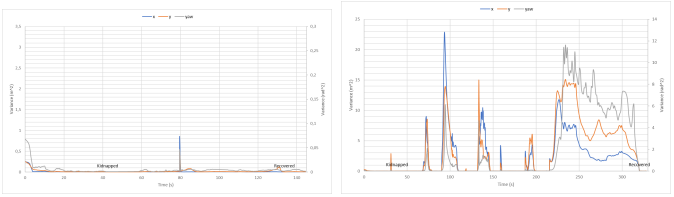
A. Localization

In this section, two experiments are presented to test the localization. Firstly, the kidnapped robot problem and the global localization are tested with two different values for the recovery behaviour constants are used to allow comparison.

In the second experiment the error on the localization is calculated by comparing the robot's estimated localization to a ground truth localization given by the Motion Capture System (MCS) that uses 12 OptiTrack cameras.

B. Recovery from Kidnapping situations

In this experiment, the robot would start with an estimate of the position, navigate remotely operated for a while, then the odometry is disabled and the robot is moved for some meters. After kidnapping the robot, the odometry is turned back on and the robot is driven remotely again until it recovers its localization. The instants where the robot is kidnapped and when it recovers its position are tagged in the graph by a label.



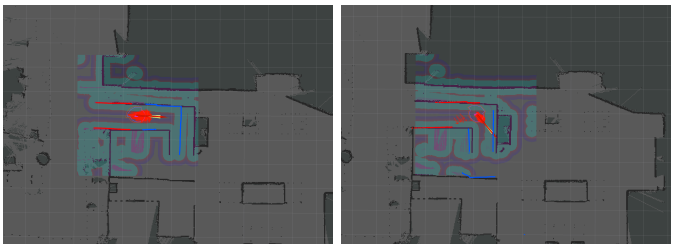
(a) $\alpha_{slow} = 0.001$. **(b)** $\alpha_{slow} = 0.01$.

Fig. 9. Variances of x, y and yaw over time in ISR's testbed for 2 values of α_{slow}.

The graph of Fig. 9(a) shows a system that is mostly confident of its localization even after getting kidnapped. There is only a moment where the variance gets very high when the robot is wrongly localized. From Fig. 9(b) it is possible to see that a higher α_{slow} when the α_{fast} is constant originates a bigger uncertainty when the robot is wrongly localized, attempting to relocalize more often.

What can be concluded is that for a lower value of α_{slow}, the robot will remain more confident of its localization even when it is not correct, though both can recover from the kidnapped robot problem. This means that the robot will trust more on its localization and have a smoother navigation.

In Fig. 10 a process of relocalization is shown.



(a) Robot wrongly localized before relocalization.

(b) The algorithm recovers the localization by having most particles around the true localization.

Fig. 10. Images of relocalization

C. Ground Truth Comparison

This experiment will determine the error in localization when using the MCS as ground truth. The ground truth of this system is not perfect since the map of the robot needs to be exactly with the same rotation as the calibration of the cameras, and that the reflecting markers need to be set up in the robot's head to be seen throughout the testbed. Since the head has some movement in relation to the base of the wheels, this adds an aditional error. With these limitations of the ground truth system, the experiment designed was a waypoint navigation to three waypoints around the testbed while recording both the robot's estimation of its location and the ground truth from the MCS.

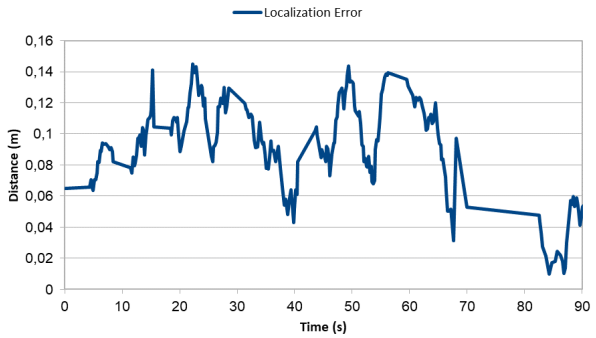


Fig. 11. Localization error from the robot's estimated pose and the ground truth from MCS

In Fig. 11 the error between the robot's estimated pose and the MCS's ground truth is shown. It is possible to see that the error is always bellow 0,15 meter and maintained an average value of 0,09 meter error throughout the 90 seconds experiment. Considering the map has a resolution of 0,025 meter and the limitations of the MCS the values represent a good result. In Fig. 12 the poses of both the robot's localization algorithm and of the MCS's ground truth over time. It is possible to see that throughout the performed navigation the error was reduced except when the robot was close to the bedroom.

Fig. 12. Localizations over time of the ground truth (blue) and the robot's estimated pose (red)

D. Waypoint Navigation

This section will present the result of two experiments. The first is part of a run of ERL Smart Cities' episode 3, where the robot has to visit several waypoints to complete its tasks.

This episode was named "Deliver coffee shop orders". At this challenge, the robot assists people in an arena resembling a coffee shop by checking the state of the tables, taking orders, bringing them to the customers and guiding a new client to a free table. Videos from this challenge can be seen in YouTube[2].

The second one is a small experiment in the ISR domestic robot testbed that shows how the robot avoids obstacles

E. ERL Smart Cities Challenge

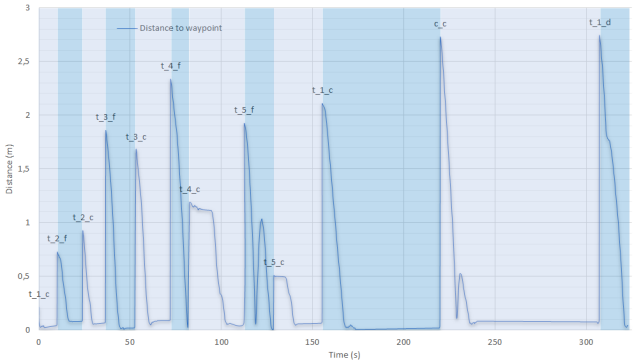


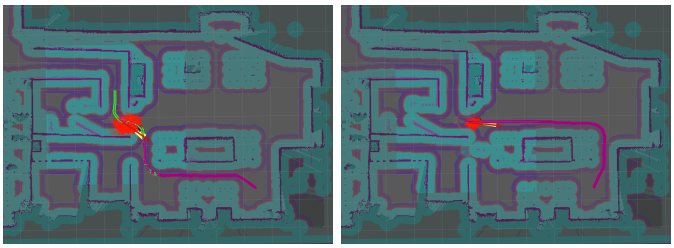
Fig. 13. Distance from the perceived robot location to the desired waypoint in meters at the ERL Smart Cities challenge

In Fig. 13, the navigation to waypoints in the coffee shop is shown in a graph that represents the distance to the desired target at each time instant as perceived by the robot. Between each color change there is a transition of desired goal and a label that indicates which waypoint has been selected.

This experiment shows that the navigation stack can travel several waypoints consecutively in a cluttered space reaching

[2]https://youtu.be/xZwMvkZsw4Q

the goal with the selected precision of 0.1 meters. In all waypoints, the robot only stopped when it was less than 0.1 meters away from the goal.



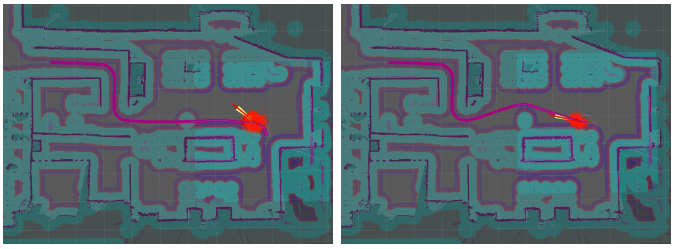
(a) Original path **(b)** Replanned path

Fig. 14. Images of the path replanning for the obstacle avoidance experiment when the path created is impossible to follow and a new path needs to be generated

1) Obstacle Avoidance: In this experiment, two navigation waypoints are sent as goal to the navigation stack. Firstly, a waypoint close to the bottom right corner of the map is sent. Since this waypoint is close to a plant decoration it will be henceforth denominated as Plant. On the way to this goal, a person blocks the path of the robot forcing it to choose another path to the goal. After reaching this goal, the robot receives another goal in the Entrance waypoint and on the way to it a person will stand in front of its path so it has to avoid him/her.

To analyse the path planning, some images of the path in the used map are presented in Fig. 14 and Fig. 15. The robot follows the path until it detects through laser scans a person blocking its path just after the frame shown in Fig. 14(a). After detecting the person, the robot replans a way to the goal through another route. This new plan can be seen in Fig. 14(b).

Some time into the second path for the *Entrance* waypoint, in Fig. 15(a), the robot turns a corner and finds an obstacle in its path which makes it replan its global path around this found obstacle, as shown in Fig. 15(b).



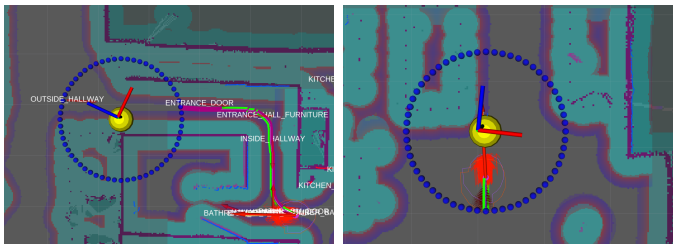
(a) Original path **(b)** Path after detected obstacle

Fig. 15. Images of the path replanning for the obstacle avoidance experiment when there is room for the robot to go around the obstacle

This experiment effectively shows that the navigation stack can properly replan in case an obstacle is perceived by the robot if another path is available.

F. People Follower

To test the people following mode, the robot was activated with that mode and a person walked at a normal pace in front of the robot. It is important to note that the people tracking and detector is not being evaluated and so there is no other person testing the resilience of the tracker. What is important to test is that given the correct position of a person, the dynamic goal component works correctly in choosing the correct goal, choosing a viable path and avoiding obstacles in the navigation towards the selected goal.



(a) Goal choice process on a corridor. **(b)** Goal choice process when the person is closer than the target distance.

Fig. 16. Selection of goal for people following

In the images presented in Fig. 16, two goal choosing results are presented. The robot position is presented as its footprint and/or the cluster of red pose particles, the position person to be followed is shown by a yellow sphere and a tf frame and the poses at 1,2 meters to the considered target are printed as a small blue sphere.

In Fig. 16(a), the robot tracks the person through a corridor with a corner which walls are 1 meter high. This allows the robot to see the person on the end of the corridor and calculate the best possible goal to get closer to the person. The robot is also able to follow the path on this narrow corridor without hitting any walls.

In frame 16(b), the person gets close to the robot which makes it plan a path backwards towards the desired distance of 1,2 meters.

G. Guiding

In this section, an experiment was conducted to test the guiding mode. This task was performed in ISRoboNet@Home tesbed where the initial position is inside the testbed behind the sofa. Then the robot goes to the *Entrance* waypoint to guide a new person inside the testbed towards the dining table. When it reaches the *Entrance*, it starts guiding the tracked person towards the given goal but trying to maintain a distance of 2 meters between itself and the tracked person. If the distance becomes greater than this threshold, the robot will stop and wait for the person to get closer, using speech to warn him/her that it is waiting.

In the plot of Fig. 17, the distance of the robot to the current goal and to the tracked person are plotted. The plot has two shaded areas, the left one represents the navigation towards the waypoint *Entrance* and the right side the guiding navigation towards the *Dining table*.

The robot sees the person and starts tracking him/her before reaching the first goal, so it started guiding immediately. The

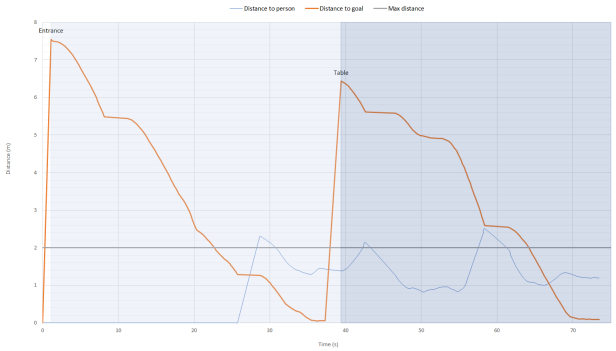


Fig. 17. Plot that depicts the distance to the person to be guided and the distance from the robot to the goal at each time instant.

robot started moving towards the table but the person stayed in the same place so the distance from the person to the robot increased to over 2 meters. The robot stopped and waited for the person to get closer before it started moving again. The person stopped once more before getting to the table that made the robot stop again to wait. In the end, the robot reached the final goal with the person closer than 2 meters from it.

VI. Conclusion

In this paper a complete multimodal navigation stack for autonomous service robots is presented, with waypoint navigation, guiding and people follower as the defined navigation modes. This was possible by integrating with correct configurations the state of the art algorithms and ROS packages with new implementations necessary for the correct behaviour. The most important new implementation was the dynamic goal package created that allows a simple dynamic goal navigation easily integrated with the navigation stack of ROS. Some videos of the results obtained can be seen in a YouTube playlist[3].

To improve the dynamic goal component a path planner could be called from the robot to the target pose and from that path, trace back the path until it finds the first point that is at the desired distance from the target. This approach has some limitations, for example, when the robot is closer to the target than the desired distance, this method would not produce a path to make the robot back some distance. A combination of the methods can perhaps produce a better solution. To exactly follow the target a Bayesian filter could be used to save the target path information and then use that information in the path planning.

References

[1] Z. Teresa, "History of service robots," in *Robotics: Concepts, Methodologies, Tools, and Applications*. IGI Global, 2014, pp. 1–14.

[2] P. U. Lima, C. Azevedo, E. Brzozowska, J. Cartucho, T. J. Dias, J. Gonçalves, M. Kinarullathil, G. Lawless, O. Lima, R. Luz *et al.*, "Socrob@ home," *KI-Künstliche Intelligenz*, pp. 1–14, 2019.

[3] "Sciroc episode 3," Oct. 2019. [Online]. Available: https://sciroc.eu/e03-deliver-coffee-shop-orders/

[4] P. U. Lima, "Motion planning and guidance," University Lecture at Instituto Superior Técnico, 2018.

[5] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[6] D. Fox, W. Burgard, and S. Thrun, "Markov localization for mobile robots in dynamic environments," *Journal of artificial intelligence research*, vol. 11, pp. 391–427, 1999.

[7] P. Zarchan and H. Musoff, *Fundamentals of Kalman filtering: a practical approach*. American Institute of Aeronautics and Astronautics, Inc., 2013.

[8] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *ICRA*, vol. 2, 1999, pp. 1322–1328.

[9] D. Fox, "Kld-sampling: Adaptive particle filters," in *Advances in neural information processing systems*, 2002, pp. 713–720.

[10] J.-C. Latombe, *Robot motion planning*. Springer Science & Business Media, 2012, vol. 124.

[11] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[12] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[13] M. Pittner, M. Hiller, F. Particke, L. Patino-Studencki, and J. Thielecke, "Systematic analysis of global and local planners for optimal trajectory planning," in *ISR 2018; 50th International Symposium on Robotics*. VDE, 2018, pp. 1–4.

[14] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[15] N. Wojke, R. Memmesheimer, and D. Paulus, "Joint operator detection and tracking for person following from mobile platforms," in *2017 20th International Conference on Information Fusion (Fusion)*. IEEE, 2017, pp. 1–8.

[16] C. Dondrup, N. Bellotto, F. Jovan, M. Hanheide *et al.*, "Real-time multisensor people tracking for human-robot spatial interaction," 2015.

[17] H. Zender, P. Jensfelt, and G.-J. M. Kruijff, "Human-and situation-aware people following," in *RO-MAN 2007-The 16th IEEE International Symposium on Robot and Human Interactive Communication*. IEEE, 2007, pp. 1131–1136.

[18] E. T. Hall, "A system for the notation of proxemic behavior 1," *American anthropologist*, vol. 65, no. 5, pp. 1003–1026, 1963.

[19] A. Leigh, J. Pineau, N. Olmedo, and H. Zhang, "Person tracking and following with 2d laser scanners," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 726–733.

[20] M. Kobilarov, G. Sukhatme, J. Hyams, and P. Batavia, "People tracking and following with mobile robot using an omnidirectional camera and a laser," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE, 2006, pp. 557–562.

[21] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2009.

[22] T. Foote, "tf: The transform library," in *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, ser. Open-Source Software workshop, April 2013, pp. 1–6.

[23] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit *et al.*, "Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," in *IJCAI*, 2003, pp. 1151–1156.

[24] "Ros wiki - move base," Oct. 2019. [Online]. Available: http://wiki.ros.org/move_base?distro=kinetic

[25] "Ros wiki - global_planner," Oct. 2019. [Online]. Available: http://wiki.ros.org/global_planner

[3] https://www.youtube.com/playlist?list=PLx27vcp5Dh_fdHwDDuN-_QPCwCOhx2GLH