The FKIE Robot System for the European Land Robot Trial 2011

Michael Brunner, Frank Höller, Achim Königs, Timo Röhling, Frank E. Schneider, Alexander Tiderko, Dirk Schulz, Dennis Wildermuth

Fraunhofer Institute for Communication, Information Processing and Ergonomics (FKIE), 53343 Wachtberg, Germany
robotik-group@fgan.de

Abstract—This paper presents technical details of the robot framework that is used by the FKIE team for the participation in the three scenarios Mule, Approach, and Camp Security of the ELROB 2011. Our navigation software used for both, the Mule and the Approach scenario, is based on a local navigation using motion patterns. Additionally, we employ a FastSLAM mapper taking a traversability analysis and virtual 2D scans of the environment as inputs. The built map will be used to return to the start point in areas with bad GPS reception. The Camp Security software is based on a people detector using image features.

I. INTRODUCTION

The European Land Robotic Trial (ELROB) is designed to demonstrate and compare the capabilities of unmanned systems in realistic scenarios and terrains. The aim of each ELROB is to get a deep insight into the field of ground robotics by testing existing solutions in practical trials. We intend to participate in three scenarios:

1) Mule — The objective is to shuttle between two given locations as frequently as possible.
2) Camp Security — A camp site is to be guarded. Intruders shall be detected, reported, and tracked if possible.
3) Approach — A given target location has to be reached through an area with static and dynamic obstacles along with dead ends, sharp turns, road blockings and narrow passages. The intelligence information gathered on the way and at the target location has to be made available at the control station.

Although manual intervention and remote control are permitted by the rules, it is our intention to let the robots master the scenarios with the maximum possible degree of autonomy. The remainder of this paper discusses various key aspects of the robot framework that we employ to achieve our goal. The sections are roughly ordered by layers, beginning with the low-level aspects such as hardware devices (section II) and middleware (section III). Afterwards, some intermediate-level algorithms for local navigation based on motion patterns (section IV), a FastSLAM mapper taking a traversability analysis and virtual 2D scans of the environment as inputs (section V), and a people detector using image features (section VI) are presented.

II. HARDWARE

Currently, we have two iRobot ATRV medium sized robots, a Telerob Telemax robot, and one QinetiQ Longcross large wheeled robot available (see figure 1). The ATRVs are relatively slow and less capable of cross-country drives, but very manoeuvrable. On the other hand, the Longcross is a true all-terrain vehicle, but rather bulky and more likely to get stuck in narrow passages. The Longcross will be the principal robot for the ELROB scenarios, the Telemax and the ATRVs are primarily intended as supplement for Camp Security.

A. ATRV

Our ATRVs are heavily modified versions of the original iRobot ATRV-Jr with four-wheel differential drive and rFLEX control architecture. Each one is equipped with a SICK LMS 200 laser scanner, a TopCon GPS receiver, a TCM compass device, and two Panasonic WV-CS850 dome cameras. The available top speed is about 1 m/s. The internal computer that is used for data processing has a mobile 2 GHz Intel Core 2 Duo processor and 2 GB memory.

B. Telemax

The Telerob Telemax robot has four articulated tracks shaped like flippers. Optionally four wheels can be attached to the tracks increasing the maximum translational velocity from 1.2 m/s to 1.6 m/s. The robot is equipped with a Hokuyo UTM-30LN laser scanner, a TopCon GPS receiver, a TCM compass device, and a Logitech Webcam Pro 9000. The internal computer that is used for data processing has a mobile 2 GHz Intel Core 2 Duo processor and 2 GB memory.
C. Longcross

The Longcross is an experimental platform weighing about 340 kg with a payload capacity of at least 150 kg. The compartment consists of carbon-fibre and is environmentally shielded. Our version is equipped with a Velodyne Lidar HDL-64E S2 3D laser scanner, an Oxford Technical Solutions Ltd RT3000 combined GPS receiver and inertial unit, and a 360 degree panorama camera with eight high-quality CCD boards. The software runs on a dedicated notebook with an Intel Core 2 Extreme Quad processor and 8 GB memory.

III. THE ROSE FRAMEWORK[1]

With robot systems leaving the laboratory environment and facing rough outdoor scenarios, the aspect of wireless communication is getting more important. Existing robot middleware does not regard the challenges of wireless communication, including aspects like fluctuation of link delay, bandwidth and reliability. The communication mechanism of a multi robot system must be capable of managing these challenges. Another issue is efficient point to multipoint (multicast) communication. This allows the availability of the same data to multiple recipients without an unnecessary high increase of network load compared to multiple unicast connections.

We therefore implement a robust multicast communication framework for a multi robot system based on wireless communication. This framework is a robot middleware, which can benefit from multicast communication optimizations of lower network layers and assumes unreliable connections. It can also cope with complete network separations, even during the startup of a robot system by using a decentralized configuration approach. A compact message format reduces the overhead on the network, therefore decreasing load and congestion possibilities.

A. Service Architecture

A RoSe service is an independent, multi-threaded standalone application. It should be kept simple, clear and maintain all-purpose functionality. The communication among the services is restricted to network communication. As an exception, inter-process communication between services on the same host can be handled by shared memory to allow exchange of large data volumes like 3D laser scans. Additionally to the data communication, there are special maintenance messages, which allow starting, stopping and terminating a service. Furthermore, a service can be configured via network.

To apply a configuration over the network and to make use of the other maintenance functionalities, the communication has to be available, before a service is configured and started. Therefore the creation of a service is performed by RoSe. For this purpose a unique and valid SID must be known right from the beginning. The SID is used to create an UDP socket used for receiving unicast messages and for sending unicast and multicast messages. For receiving messages an additional thread is used. After the communication object is created and initialised it is assigned to the service, allowing the service to be either configured manually or remotely. The configuration proceedings of a service are described in section III-B. After the configuration is completed, the service can be started and stopped manually or via the network.

Due to the remote starting functionality of RoSe services, each application used in RoSe is based on a basic service class. This class does not allow a service to perform the starting process before the service was initialised with a configuration or explicitly without a configuration. While in the configuration stage, the application may read the assigned configuration and add its own initialising routines, e.g. to join one or more multicast groups for receiving published data from other services. For each multicast group a new multicast socket and an additional thread for packet reception is created and added to the service’s communication object. Other stages, where an application service can add its own routines are the starting, stopping, terminating events and on receiving a message through one of the above-mentioned sockets.

The plainness of our service concept may lead to problems if two services have to use one device. An example is a pan and tilt camera with a zoom lens that provides two services; one for controlling the movement and the other for controlling the optical functions. However, both of the services transmit different types of command via the same serial port. Another difficulty appears when regarding a robot simulation service: Here all artificial sensor data is created by only one service, and due to the limitation to one unicast socket and therefore one LSID, complete message identification by the receiving services is not possible. For this reason virtual services are introduced. The behaviour of virtual services is identical to normal services. However each virtual service has its own thread and its individual LSID for communication. It is started and controlled by a master service on the same host. Since the virtual services are part of the master service’s application, they and their functionality are controlled by the master service, rendering additional network communication between multiple services unnecessary. The drawback of virtual services is, that they are handled in the same process space, so that all virtual services may fail due to one bug in a single virtual service.

B. Configuration

The RoSe framework provides flexible configuration options for its services. The configuration is described in an XML format and provides all needed parameter types e.g. SID, integer, float, boolean and string. The configuration is read and parsed by the RoSe framework and is made available to a service at the beginning of its initialising process. The configuration can be illustrated with a configuration tree. An example for such a tree is shown in figure 2. Regarding a single service only a subtree of the configuration tree is used.

There are multiple possibilities to configure services in RoSe. The first one is to configure each service separately. In this case the configuration of each service is stored in a separate XML file. When it comes to boot up a whole robot system, a lot of services are involved, leading to a multitude of configuration files, which all together are hard to manage. Therefore two alternatives, the Service Manager and the Config Manager, are introduced. The Service Manager
is used to manage the services on one host and the Config Manager is used to control all services in network.

1) Single Service: The configuration of a service contains all parameters used by this service and the parameters of all virtual services which are controlled by this service. If a single service is configured, an XML configuration file is passed to the service on the command line. The service is then initialised directly after its creation. The drawback of this method is that multiple configuration files must be managed if a whole robot system is started. Alternatively a service can be created and then initialised through the network. Using a special message the configuration is serialised and sent to the service. The service initialises directly after the reception of this configuration command. To use this alternative configuration scheme no XML file is passed to the service. The network configuration may also be used to reconfigure an already running service. Every service should be able to be started with an empty configuration; in this case a default configuration should be used.

In all of the above cases a service requires a unique SID to be created. This is an inherent and constant part of a service’s parameter. This parameter is necessary to initialise the network socket of the service and, if the configuration is sent through the network, to listen for the service configuration. As a consequence the LSID must be available while the service is created and cannot be stored in the configuration.

2) Decentralized Configuration: In the decentralized configuration approach a Service Manager is used to control all services on a single host. It is configured and started as a normal RoSe service. The configuration of the Service Manager contains the configuration of all of its controlled services. During the initialisation of the Service Manager it creates all controlled services and sends them their configuration. Therefore the special configuration message used, which is restricted to the corresponding configuration subtree of the service. The decentralized configuration approach is illustrated in figure 3. The services should be configured using a service manager but a single service configuration without a Service Manager is still possible.

The advantage of the decentralized configuration approach is that each host can be configured without being dependant on other hosts. On the other hand a configuration file for each host has to be maintained.

3) Centralized Configuration: In the centralized configuration approach an additional service is introduced. This service is called the Configuration Manager. Here the services of each host are still controlled by a Service Manager, but the Service Manager is configured by the Configuration Manager via the network. This allows for the whole robot system using only one configuration file. The initial configuration is passed to the Configuration Manager. If necessary the configuration manager may change some configuration parameters and then send configuration messages to the Service Managers containing the specific configuration subtree.

The disadvantage of this approach is that each host to be configured must be available on the network. Under the assumption that connections are not reliable, the decentralised approach should be favoured if possible, in order to ensure that robot systems obtain their full configuration in a timely manner. However, in controlled and test environments it may be helpful to use the centralized approach. The concept of this approach is illustrated in figure 4.

C. Message Concept

The main goal of the message design in RoSe is simplicity and clarity. Furthermore, the available messages should be reusable in multiple services, e.g. the position message is used by robot and camera movement control service to tell other services their current position. The sender’s SID identifies the source of the position information and therefore the context of the information. So the coordinates stored in a message sent by the robot movement control service describes the robot’s
current position in the environment and the same coordinates transmitted by a camera movement control represent the position and orientation of the camera on the robot. Reusing messages demands a standardised unit format. Therefore, all RoSe services are obliged to satisfy the International System of Units (SI).

In addition the messages must be compact to reduce the overhead on the network. So each new message in RoSe must implement optimized serialize and deserialize methods. Of course all objects used in a message, except basic data types, should implement a special interface to allow serialization and deserialization and to avoid implementing these methods in every message that uses these objects. Every new message, which is created, must be registered with a unique ID number. On the basis of this ID the message is identified in the network and the appropriate deserialization method can be chosen. This allows an easy deserialization in different programming languages, in our case C++.

For controlling services on the network a special message is introduced. It allows initializing, starting, stopping and terminating a service. The initialisation option of this message is used by the Service Manager and the Configuration Manager to configure the services as described above.

IV. Collision Avoidance and Local Navigation[2]

In the design of robot systems operating in unstructured outdoor environments, special care has to be taken that the robots do not accidentally collide with obstacles in their vicinity. This task is exacerbated by different ground surfaces which have a distinct effect on the wheel grip. The resulting deviation has to be anticipated to ensure the reproducibility of motions and thus making collision avoidance possible.

Additional complications arise from robots which were designed for remote-control, like the Longcross robot. Such robots are normally equipped with relatively simple motor controllers, which normally cannot process the velocity commands generated by classic navigation algorithms.

In this section we discuss our approach which allows a mobile robot with any kind of electronic motor controller to operate in cluttered outdoor environments. To be able to improve the navigation behavior of the robot in unknown locations, our approach follows a local navigation paradigm and does not need a map of the environment. Instead, the robot’s motion control decides solely based on the robot’s sensory input.

The motion planning for the robot is based on a tree-search technique which we developed to suit the special requirements of the robot’s motor controllers. Our planning algorithm composes trees by combining predefined motions and extracts a path towards the target coordinate. To tackle the surface traction problem, the robot’s movements are measured on the fly. The collected data is used to update the measured movement part of the Motion Patterns. The upgraded Motion Patterns are handed over to the planning process and used for the tree generation.

Waypoint navigation is one of the fundamental tasks for autonomous mobile robots. Many popular systems utilize a navigation technique which defines a limited set of commands or command-sequences and greedily decides in every computing interval which entity should be applied. In [3] a path generator uses combinations of speeds and steering angles to generate trajectories which are then further evaluated. Therefore, an occupancy grid generated by a 3D-laser and different weighting functions is used. This is similar to the Dynamic Window Approach (DWA) [4], [5] because both methods restrict the search to a single time step, i.e. they select the next best controls based on the current sensor input and a model of the robot’s dynamics.

The concept of motion template based learning has been previously employed to simplify the learning of complex motions [6]. In contrast to our approach the templates have parameters which are adjusted to fit the desired trajectory. This implies a usable correlation between parameter input and drive-train behavior. Instead of trying to adopt the local navigation online, future robot positions can be estimated by regarding the current terrain type [7]. Therefore, inertial sensor data is processed with Gaussian process models to determine the movement velocities for position estimation and to deduce the terrain type from vibrations. This method is very effective if all surfaces are known beforehand.

A. Using Motion Patterns for local Navigation

1) Motion Patterns: The core of the overall approach is a local navigation planning component which directly controls the robot and steers it on a collision-free path from its current position to a given destination. This destination is defined by a coordinate and a distance threshold, which is a widely used technique to define a target area [8].

Since remote controlled robots lack a velocity regulator circuit, the commands influence the motor power directly, so their outcome depends on many factors and is far too expensive to compute. Thus we introduce Motion Patterns to simplify the motion planning. The first component of each Motion Pattern is a series of robot control commands. These commands can be of any type; when used with a Qinetiq Longcross robot, they are motor power commands. These control commands are immutable, which implies that the overall velocity of a Motion Pattern cannot be adjusted. The second component of a Motion Pattern is an array of oriented positions. It represents the trajectory on which the robot will probably move when the command series is sent to the robot. This is a popular method, because it “allows computing the cost of a motion without explicitly considering the motion itself” [9]. These trajectories can be combined to form longer paths and checked for collisions e.g. using an occupancy grid as presented in [2]. When the Motion Patterns in such a path are reduced to their robot control commands, a path is only a large sequence of these commands, and thus can be executed by the robot sequentially. Notice that the number, shape, and complexity of Motion Patterns are not restricted, but definitely have an impact on the later described planning process. To combine Motion Patterns to a continuous path we have to make sure that the transitions between the chosen patterns are smooth. To accomplish this, the initial and final velocities are
Motion Learning

2) Path Planning: Based on the model above, we can now build a collision-free tree of Motion Patterns \( T = (V,E) \) consisting of nodes \( V \) and possible transitions \( E \) between nodes. Every node \( V \) represents a Motion Pattern. The root node is defined by the final state of the currently applied Motion Pattern. New nodes are created in a breadth-first manner and are connected to their parent node if they are applicable, inter alia collision-free, at this position.

After adding a node it is rated with a weight-equivalent: Since the planning algorithm intends to find the fastest path to a given destination, distances from the root to other nodes are represented by the approximated travel time. Please consult [2] for a detailed description of the used heuristic. During the construction of the tree the most promising node is marked, because it represents the best path found so far. The creation of new nodes is aborted when a sufficient tree depth is reached or a time limit is exceeded. A potential new path is available after a tree has been constructed.

To be certain that a new path is available in time, the tree construction has to be finished before the Motion Pattern which is currently applied by the robot, is executed completely. To ensure this, the time limit for the tree construction process is equal to the time consumption of the quickest Motion Pattern available. As mentioned before, the size of the Motion Pattern pool has a significant impact on the generated tree: While more available patterns enhance the quality of the resulting tree, they also decrease the tree depth that can be reached in the computation time window. A suitable tree size has to be chosen carefully. An example tree from our current implementation can be seen in figure 5.

Using the previously marked node, a series of Motion Patterns representing a path towards the destination can be extracted from the tree.

B. Motion Learning

A problem arising from our special kind of local navigation is its sensitivity to surface and traction changes. Motion Patterns are created for specific surfaces only. And it is unlikely that the surface or the surface’s condition always remains constant, especially in outdoor scenarios. To compensate this, a basic learning mechanism has been added: While the command set of a Motion Pattern is executed, the robot measures its movements, i.e. its relative position and orientation. The measurements are integrated in the Motion Pattern’s existing prediction using a component-by-component exponential smoothing function to allow continuous learning:

\[ \bar{m}_{p,t} = (1-w)m_{p,t} + (w)\bar{m}_{p,t-1} \] (IV.1)

with \( 0 \leq w < 1 \). Here \( m_{p,t} \) is the measured trajectory of a Motion Pattern \( p \) at time \( t \), \( \bar{m}_{p,t-1} \) represents the existing prediction of the pattern and \( \bar{m}_{p,t} \) the updated prediction. To limit the impact of new measurements, \( w \) should not be larger than \( \frac{1}{2} \). Note that it is not possible to adapt the command sequence to match the desired trajectory, because the mapping from trajectories to commands is unknown and possibly not even computable. Figure 6 depicts how the learning mechanism is integrated in the navigation process.

A side effect of the Motion Pattern adaption is the possibility of pattern pool depletion: It occurs when an update causes one or more patterns to become best suited for a specific maneuver that was previously covered by a third pattern. As this third Motion Pattern will not be used any more, it cannot be updated and remains unused. In order to counteract this effect, we delay pattern updates until new motion data for all Motion Patterns has been collected. Then, we update all patterns at once. The drawback of this method is that the trajectory construction will be less accurate due to outdated Motion Patterns.

Between the pattern updates some Motion Patterns will be executed more frequently than others. Therefore, we apply a secondary exponential smoothing with a much larger \( w \) which tracks the amount of change that is to be applied with the next update. This technique effectively balances the disproportionate impact that more frequently used Motion Patterns have on the pattern pool.
V. SIMULTANEOUS LOCALISATION AND MAP BUILDING[10]

Especially in the Approach scenario, when the robot must navigate through unknown terrain to a given GPS location, the ability to gather knowledge about the environment is crucial. Unlike airborne vehicles, which may simply home towards the desired position, ground vehicles have to keep track of obstacles and dead ends to navigate successfully. Therefore, we employ SLAM techniques in combination with GPS to build an accurate map of the robot’s surroundings, and use the map to compute the shortest path to our destination. Although the Approach scenario is the primary use case for SLAM, other scenarios benefit from the improved map localization, particularly where the robot passes through terrain with poor or intermittent GPS reception.

The core algorithm for our map building is a particle filter for 2D FastSLAM. The remainder of this section gives an overview of the relevant components.

A. Virtual 2D scans from 3D data

Although the robot has sufficient sensor hardware to capture 3D range data fast enough for real time applications, the scenarios are basically 2D, disregarding the occasional hill or hole. As 2D SLAM is much faster than its 3D counterpart, it is prudent to reduce the massive amount of range data to a more manageable size, as long as all or at least most distinctive features in the environment are preserved.

One major disadvantage of simple 2D laser range finders is their dependence on the robot’s roll and pitch angle. Even slight bumps in the road may cause the scan plane to shift considerably, rendering consecutive scans difficult or even impossible to correlate. A full 3D scan can be used to extract the ground plane and infer the current angular orientation, and therefore is much less susceptible to the aforementioned problem. However, if the ground is not completely flat, a simple virtual scan plane above ground may still result in false positives, e.g. if the robot stands at the foot of a hill.

Thus, we exploit the geometric configuration of the Velodyne device, which provides 64 concentric scan rings, and evaluate the smoothness of these rings in each scan point to infer the traversability of the ground. Further, we use the normal in each scan point, computed by cross product from adjacent points, to eliminate ascents or descents which are too steep for the robot.

The classification of all scan points into traversable and non-traversable points enables our software to construct a virtual 2D scan using raytracing. The first non-traversable point that is encountered during raytracing from the robot center is considered as hit point for the virtual 2D laser. The resulting 2D scans are largely independent of the robot’s roll and pitch angle and eligible for scan registration.

B. Scan Registration

1) Terminology: Unless mentioned otherwise, the variables $p$ and $q$ denote non-oriented points in $\mathbb{R}^n$. An oriented relative pose $\delta$ designates a translation vector $x_{\delta} \in \mathbb{R}^n$ and a rotation $\rho_{\delta}$. Rotations in $\mathbb{R}^2$ are single orientation angles, rotations in $\mathbb{R}^3$ consist of three angles and are best represented by unit quaternions. Pose transformations are defined as in Lu and Milios [11], in particular

$$\delta \oplus p := \rho_{\delta}(p) + x_{\delta}, \quad (V.1)$$

which transforms a relative position $p$ with respect to a given pose $\delta$ (see figure 8).

2) Problem description: Let $S_i \subset \mathbb{R}^n$ and $S_j \subset \mathbb{R}^n$ denote two $n$-dimensional scans in a local frame of reference and $\delta_{ij}$ the relative position and orientation of $S_i$ with respect to $S_j$. Assuming that there are two non-empty subsets $F_i \subset S_i$ and $F_j \subset S_j$ of features that overlap, there is a mapping $\pi_{ij}$ that assigns each scan point $p \in F_i$ to its counterpart in $F_j$, so that

$$\delta_{ij} \oplus p = \pi_{ij}(p). \quad (V.2)$$
In practice, the robot’s odometry provides only a coarse estimate \( \delta_{ij} \) which does not satisfy the above equation. The introduced registration error can be quantified as

\[
\varepsilon_{ij}^2(\delta) = \sum_{p \in F_i} ||\delta \oplus p - \pi_{ij}(p)||^2, \tag{V.3}
\]

which is sought to be minimised. For known \( F_i \) and \( \pi_{ij} \), closed-form solutions exist [12] that optimise \( \delta_{ij} \) with respect to \( \varepsilon_{ij} \).

However, neither the subsets \( F_i \) and \( F_j \) nor the correspondence mapping \( \pi_{ij} \) is known in advance. In fact, a bijective mapping as asserted by equation V.2 may not even exist due to the type of range sensors usually employed (see figure 9). Thus, \( \pi_{ij} \) is approximated and the optimisation process is iterated until \( \pi_{ij} \) and \( \delta_{ij} \) converge towards \( \pi_{ij} \) and \( \delta_{ij} \) respectively, usually by employing a standard algorithm such as Hillclimbing or ICP.

As is shown by Rusinkiewicz and Levoy [13], the nearest neighbour heuristic

\[
\pi_{ij}(\delta, p) = \arg \min_{q \in S_i} ||\delta \oplus p - q|| \tag{V.4}
\]

for \( p \in S_i \) proves to be one of the most robust approximations. Unfortunately, the described technique is quite sensitive to the initial estimate of \( \delta_{ij} \), so that the scan registration may fail to converge even for moderate estimation errors. This is due to the complex structure of the error function \( \varepsilon_{ij} \), which tends to have many local minima when applied to real-world sensor data.

In order to enhance the convergence radius, the influence of local minima has to be mitigated. This can be achieved either with a smoothed error function, e.g. by applying a convolution filter, or with an improved initial pose estimate. However, the former decreases the accuracy of the scan registration considerably, and the latter requires at least an approximate solution to the scan registration problem itself. Therefore, we employ a recursive solution based on deterministic annealing that uses a cluster-based scan registration as substitute for p-alignment to improve the robustness of the optimisation.

C. Clustering and Annealing

Laser scans of outdoor environments are inherently hierarchical. Large-scale entities such as buildings and open spaces contain smaller objects such as cars or trees. Depending on the scan resolution, this hierarchy may well extend down to leaves or wall bricks. Obviously, it does not make sense to match these fine structures if the estimated position is still off by a dozen metres. This section introduces the necessary data structures and algorithms to deal with different levels of details during scan registration.

1) Cluster definition: Let \( S_i \subset \mathbb{R}^n \) be a scan. A cluster \( c_r \subset S_i \) is a grouping of scan points which lie close together, i.e.

\[
\forall p, q \in c_r : ||p - q|| \leq 2r. \tag{V.5}
\]

A cluster set \( C_{r,1} \) for a scan \( S_i \) is a minimal set of non-empty clusters \( \{c_{r,1}, \ldots, c_{r,t}\} \) that assigns each point \( p \in S_i \) to a unique cluster, i.e.

\[
\forall p \in S_i \exists c \in C_{r,1} : p \in c \tag{V.6}
\]

and

\[
\forall c, c' \in C_{r,1} : c \cap c' = \emptyset \lor c = c'. \tag{V.7}
\]

Clusters may consist of a single point, thus \( S_i \) itself can be interpreted as a cluster set \( C_{0,i} \).

2) Scan clustering: Let \( r \) be the cluster radius and \( C_{r,i}, C_{r,j} \) cluster sets that are derived from the scans \( S_i, S_j \) respectively, and let \( c_i, c_j \) be two clusters from \( C_{r,i}, C_{r,j} \). Assuming that the clusters can be represented as Gaussian distributions with means \( \mu_i, \mu_j \) and standard deviation \( \sigma \), the likelihood of both clusters being equivalent is

\[
P(c_i \equiv c_j) = \eta \exp \left( -\frac{||\mu_i - \mu_j||^2}{2\sigma^2} \right), \tag{V.8}
\]

with \( \eta \) as normalisation factor. The value of \( \sigma \) is chosen so that a sampling of the distribution has 99 per cent of its samples lie within the radius \( 3\sigma = 2r \), i.e. \( \sigma = \frac{2}{3}r \). Omitting \( \eta \), switching to log-likelihoods, and taking the square root yields the distance function

\[
d_r(c_i, c_j) = \frac{3}{2\sqrt{2}\sigma} ||\mu_i - \mu_j||. \tag{V.9}
\]

The function \( d_r \) allows clusters to be interpreted as simple points. Therefore, the mapping \( \pi_{ij} \) between two scans can be adapted to cluster sets by

\[
\tilde{\pi}_{r,ij}(\delta, c_i) = \arg \min_{c_j \in C_{r,j}} [d_r(\delta \oplus c_i, c_j)] \tag{V.10}
\]

As outliers may occur, especially in situations where no bijective mapping exists (figure 10), the distance function \( d_r \) is discounted exponentially, yielding a score function

\[
s_{r,ij}(\delta) = \sum_{c_i \in C_{r,i}} \exp \left( -d_r^2(\delta \oplus c_i, \tilde{\pi}_{r,ij}(\delta, c_i)) \right) \tag{V.11}
\]

that is to be maximised.
D. Deterministic Annealing

In order to achieve both robust convergence and high scan precision, a deterministic annealing technique is applied that reduces the cluster size \( r \) gradually. The motivation for this approach stems from the codebook clustering problem which is defined as follows. Given a set

\[
X = \{x_1, \ldots, x_n\}
\]

of vectors and a codebook size \( k \ll n \), find a set

\[
V = \{v_1, \ldots, v_k\}
\]

of codebook vectors that minimises the distortion

\[
\varepsilon^2 = \sum_{x \in X} \min_{v \in V} \|x - v\|^2.
\]

The classical simulated annealing optimises \( \varepsilon \) with respect to the full codebook size, i.e. all \( k \) codebook vectors are initialised randomly and then optimised. Local minima are avoided by a randomised walk along \( \varepsilon \) that accepts a temporary deterioration of \( \varepsilon \) with a probability that slowly converges towards zero. While this method is known to find the global minimum, the required time is potentially high, especially for large numbers of local minima.

In contrast, the deterministic annealing does not randomise the optimisation itself, but relies on a fuzzy probabilistic assignment \( P_T(x,v) \) of input vectors to codebook vectors (hence the name soft-clustering), such that

\[
\sum_{v \in V} P_T(x,v) = 1.
\]

The distortion function then becomes

\[
\varepsilon^2_T = \sum_{x \in X} \sum_{v \in V} P_T(x,v) \|x - v\|^2.
\]

The temperature \( T \) measures the fuzziness of this assignment, with \( P_0(x,v) = \delta_x,\pi(v) \) for a unique mapping \( \pi \), and \( P_T(x,v) \) uniformly distributed for \( T \rightarrow \infty \). Beginning with a high temperature \( T \), where the optimal codebook contains the mean of \( X \) only, the influence of each input vector over the codebook is gradually confined to a smaller subset of potential codebook vectors, increasing the complexity of \( \varepsilon_T \), and the size of \( V \), until the required number of codebook vectors is reached. Note that while the assignments are probabilistic, the optimisation itself is strictly deterministic.

In the context of scan registration, the notion of fuzzy assignment translates to the clustering of scan points. For very large \( r \), the scans are collapsed into their mean point. With progressive cooling of \( r \), the clusters begin to split, until each cluster contains a single scan point. At this point, the optimisation is equivalent to a non-clustered, non-annealing version, with the crucial difference that the scans are already well aligned from previous optimisation steps.

In practise, the annealing is controlled by the exponential sequence

\[
r_k = r_0 \alpha^k
\]

with \( 0 < \alpha < 1 \) and

\[
r_0 = \frac{1}{2} r_{\text{map}} \alpha^{-K}
\]

for a given number of iterations \( K \) and the final map resolution \( r_{\text{map}} \). Each iteration matches the corresponding cluster set \( C_{r_k,i} \) against \( C_{r_k,j} \).

E. Sensor Model

Each particle update involves the evaluation of the current laser scan with respect to the already existing map. For the sensor model, we used the generic laser beam model by Thrun, Burgard and Fox [14]. The model is a mixture of four probability distributions \( p_{\text{hit}}, p_{\text{short}}, p_{\text{max}}, \) and \( p_{\text{noise}} \). Each distribution accounts for a specific range measurement cause, namely a map obstacle hit, an unexpected dynamic object that does not belong to the map, no obstacle within range, and a random sensor error. For the subsequent formulas, \( z \) denotes the range of a beam that has been measured, \( z^* \) denotes the expected range given the current map, and \( \eta \) is a normalizer that ensures that the integral over the distribution is 1. The distribution

\[
p_{\text{hit}} = \eta \cdot \mathcal{N}(z, z^*, \sigma^2)
\]

is a simple normal distribution that depends largely on the range noise of the sensor itself. The distribution

\[
p_{\text{short}} = p_{\text{max}} \cdot \lambda \cdot e^{-\lambda z}
\]

models the probability of an unexpected object in the beam, which decreases exponentially with increasing distance from the laser. In our current implementation, we use \( \lambda = 0.2 \). The distribution

\[
p_{\text{max}} = \delta_{z_{\text{max}}}
\]

is a point mass distribution that models max range measurements. The distribution

\[
p_{\text{noise}} = z_{\text{max}}^{-1}
\]

is a uniform distribution that accounts for random measurements. All four distributions are mixed with different weights:

\[
P(z|z^*) = u_{\text{hit}} p_{\text{hit}}(z) + u_{\text{short}} p_{\text{short}}(z) + u_{\text{max}} p_{\text{max}}(z) + u_{\text{noise}} p_{\text{noise}}(z).
\]

For our current implementation, the weights are \( u_{\text{hit}} = 0.5 \), \( u_{\text{short}} = 0.3 \), \( u_{\text{max}} = 0.02 \), and \( u_{\text{noise}} = 0.18 \).

As we use likelihood-based occupancy grid maps, we cannot easily infer a single expected distance for a beam. Instead, we
perform a raytracing along the beam and integrate over all possible expected distances with

\[ P(z) = \int_0^{z_{\text{max}}} P(z^*) P(z|z^*) dz^*. \]  

(V.24)

VI. FEATUREBASED PEOPLE DETECTION AND TRACKING

Being able to detect and track the persons in its vicinity is an important capability for mobile robots operating in populated environments. It enables the robots to take the motions of the humans into account during its own motion planning and it is an important prerequisite for several human robot interaction tasks. For interaction purposes it is also often necessary that a robot is able to memorize specific persons and recognize them again, even if they were not visible for longer periods of time, think, i.e., of mule robots that have to service individual clients.

Vision-based people detection systems have become more and more efficient and robust over the last years. They produce useful results with few false positives on challenging datasets. Some detectors are already efficient enough to run on a mobile robot.

However, in many people tracking systems today the detectors are used to carry out tracking by detection, which means that naive detection is used in every frame and the persons’ motions are afterwards estimated by associating detections to tracks from one timestep to the next. A real distinction between the tracked persons often does not take place, so these approaches fail if persons are close together or if trajectories cross. Another disadvantage of tracking by detection is its serious waste of computing resources. Given that a person does not change its complete appearance from one frame to the next and the number of people stays constant in most frames of a video sequence, it is more efficient to re-use the gathered knowledge from previous frames to track the persons, instead of starting an uninformulated people detection every frame anew. So, for a mobile robot it would be feasible to do a complete detection only on some frames in order to detect new people and do a position update, i.e., tracking, for known people in between.

In this paper we present a people tracking system that aims at such a close integration of a state of the art feature-based object detector with tracking. For this purpose, the FastPRISM object detector by Lehman [15] is used. It is a speed up variant of the implicit shape model (ISM) technique originally developed by Leibe [16]. The tracking mechanism itself is also inspired by the original ISM method. Basically the features which contributed to the initial detection are tracked over the image sequence and vote for the new center positions. The features are tracked by performing a correspondence search for matching features in subsequent images. This way, the approach does not need to apply a motion model for persons in video images which can be brittle if a moving camera is used. If the voting of the tracked features is consistent enough, new features are added and old features which were not observed for a time, are removed. This way, the tracking process adapts to changes in the appearance of the person and therefore becomes robust against mixing up persons. Additionally it prevents the number of features from growing to large, which speeds up the tracking process considerably. In this way a set of features is kept for every person which is used to track the person through the sequence by voting for its center position.

In the remainder of this section we will first explain the people detector we utilize a bit closer in VI-A and describe how the tracking system is initialized by this detector in VI-B. The next part, VI-C, illustrates the tracking step which results in an approximation of the person’s new center position and a set of features which voted for this position. VI-D then explains in detail how this set of voting features is used to optimize position and size of the rectangle around the person. The adaption of the feature set to the current appearance of the person is subject of VI-E.

A. People Detector

The initialization of the tracking is done automatically in the proposed system by using a trained people detector. We are using a variant of the ISM detector, FastPRISM, which Lehman described in [15]. It is basically a speed up of ISM by applying the Efficient Subwindow Search (ESS) technique [17] to the Hough based ISM detector. Similar to ESS it leads to a detector that only analyzes a subset of the possible object center locations.

The detector learns a feature codebook by clustering all observed image features. For each cluster all similar features are parsed and their relative positions towards the known object center are saved. During detection the observed features are matched against the codebook and then do a Hough vote for all relative positions which were observed during training. Inside this Hough space the maximum position is found and considered as center of a person.

This combines the discriminative power of shape based models with the robustness against partly occluded objects of feature based detection methods. Silhouette based methods, for example [18] deliver good results as long as the silhouette can be reconstructed from the image data. But in many cases silhouettes are not completely visible and are distorted by partial occlusion. On the other hand solely feature based detectors like the Bag of Words detector [17] are ignoring
the information that is carried in the relative position of the features and perform worse in many cases.

The people detector implementation used, could be exchanged against every other people detector which works on image features, for example the Bag of Words detector that is described in [17] or [19].

B. Initialization

On a successful object detection the detector returns: (1) The objects center \( C_0 = (x_0, y_0, \text{size}) \) with size being a representation for the size of the person, (2) a set of \( n \) detected features \( F_0 = \{ f_i^0 = (x_i^0, y_i^0) \} \) with corresponding image positions, and (3) a mapping \( I_0 = \{(f_i^0, c_i)\} \) that maps each detected feature \( f_i \) to the codebook entry \( c_i \) it was matched against.

For the features the coordinates relative to the object center \( P_{rel} = \{ p_i^{rel} = (x_i^0 - x_o, y_i^0 - y_o) = (x_i^{rel}, y_i^{rel}) \} \) are also calculated. This information is aggregated into a tracked object data structure. We are using a scale factor \( s \) to take size changes of the person inside the image into account. Of course \( s \) is initialized to 1.0.

C. Tracking Step

In every new Image the same feature extractor as in the people detection step is applied to the image. The resulting \( m \) features \( F = \{ f_j = (x_j^f, y_j^f) \} \) are matched against the codebook of the detector, which results in the mapping \( I = \{(f_j, c_j)\} \). The codebook entries in \( I \) and \( I_0 \) are compared and for each feature with \( c_j = c_i \) a vote is casted into a vote image. The vote is casted at the position

\[
\hat{v}_{ij} = (x_j^f - x_i^{rel} \cdot s, y_j^f - y_i^{rel} \cdot s). \tag{VI.1}
\]

This results in a number \( k \) of votes with \( k \) between 0 and \( n \cdot m \). Usually the number of votes is close to \( n \). For each vote the relative point \( p_i^{rel} \) and the absolute feature position \( f_j \) is stored.

Note, that the features found will generally not vote for the same center-pixel, but the votes are distributed in the vote image around the true center. To compensate for this, we let the features cast “noisy” votes, i.e. each vote is distributed around a center-pixel following a Gaussian distribution with a standard deviation of 10 pixel. In a region with many votes these Gaussians will overlap and sum up. So the maximum in the created vote image is a good approximation for the center position \( C \). Fig. 12 shows some example vote images to illustrate this procedure.

D. Person Size and Position Estimation

The size of the person in the image changes, if the distance between person and camera changes. This has an influence on the quality of the vote maximum. If the person moves further away from the camera, the features will move closer together. As a result, the votes for the center will become more scattered, as long as the distance between the vote and feature position is not reduced. If the person moves closer to the camera, the votes get more scattered as well, because the features move further away from each other; the distance between vote and feature position should now be extended. In both cases the votes are scattered if the feature offsets from the center are not altered. This effect is partially mitigated by the noisy voting scheme. However, it improves the tracking performance considerably, if we take the scale of the person’s size into account during voting. To do this, we need to estimate the size of the person in the current image and calculate a scale factor \( s \) for the relative positions which are all calculated for persons of size \( s = 1 \) during training.

We use linear least-squares estimation to optimize the scale \( s \) and the center position \( C = (x_c, y_c) \) simultaneously. For all votes \( v_{ij} \) which are closer to \( C \) than some threshold the following noisy constraints should hold:

\[
\begin{align*}
    x_c + x_i^{rel} \cdot s &= x_j^f \\
    y_c + y_i^{rel} \cdot s &= y_j^f.
\end{align*}
\]

In other words, if we add to the center position \( C \) the relative offset \( p_i^{rel} \) scaled with factor \( s \), we should get close to the feature position \( f_j \). From these equations we construct the following system of linear equations for all inlier votes \( v_1, \ldots, v_k \):

\[
\begin{pmatrix}
1 & 0 & x_{v_1}^{rel} \\
0 & 1 & y_{v_1}^{rel} \\
\vdots & \vdots & \vdots \\
1 & 0 & x_{v_k}^{rel} \\
0 & 1 & y_{v_k}^{rel}
\end{pmatrix}
\begin{pmatrix}
x_c \\
y_c \end{pmatrix}
= \begin{pmatrix}
x_f^f \\
y_f^f \end{pmatrix} \tag{VI.2}
\]

The equation system estimates the optimal center position and person scale factor. The voting process is now merely used to eliminate outlying feature matches. The estimated value of \( s \) is used for the voting in the next image. This implies that the size of a person does not change too much from one image to the next within a sequence, which is a realistic assumption, as long as the time gap between processed images does not get too large. To some extend a larger time gap can be compensated by larger Gaussian (i.e. higher sigma) in the voting step.

E. Adding new Features

At this point we calculate a score to decide if we successfully updated the person’s position, or not. The score is calculated as

\[
\text{score} = \frac{\hat{k}}{n} \tag{VI.3}
\]

which is the number of inlying votes \( \hat{k} \) divided by the number of tracked features \( n \). The intuitive explanation for this score is, that it represents how many features voted for the same center. If this score reaches 1, there is a consensus of all features. A thresholding for the score is done to decide if the position update was successful. In our experiments a threshold of 0.3, i.e. 30% features voted for the correct center, turned out to be a good choice. 30% correct votes sounds few, but keep in mind, that not all features vote in each image. Only if the score is above the threshold the feature set and contribution scores will be altered.
To adapt the tracked object to changes in appearance and be robust against background changes, we need to add new features and remove wrong or unobservable features constantly. For each feature we remember a contribution counter. We add to this counter if the feature is observed in an image and votes for the correct center and subtract from this counter if the feature is not observed or votes for the wrong center. So this contribution counter delivers an idea on how often the feature added to the correct center, i.e. how unique it is for this tracked person and how often it is visible, i.e. how stable it is.

The FastPRISM people detector allows to determine which features add positively to a detection for a given object center $C$ very fast. We use this functionality here to let the people detector select the most promising features from the remaining feature set. That is we remove all features which voted for the correct center from the feature set and supply the remaining features combined with the fixed window of the tracked person to the detector. This results in a set of positive features, similar to the set during initialization.

We could now just add these selected features to our tracked feature set. But for performance reasons we limit the number of tracked features. So we exchange features with a negative contribution score against the most promising new features, that is we remove all features which voted for the correct center, i.e. how unique it is for this tracked person and how often it is visible, i.e. how stable it is. This leads to a set of features which is best adapted to the current appearance of the person and allows the algorithm to still be fast. Allowing more features can help to re-identify a person after long times of occlusion, but it also reduces the overall score, because then, a higher number of features is not observed in a given frame.

VII. CONCLUSION

In this paper, we presented the technical and algorithmic basics of our robot system that will participate in the ELROB 2011. The core of our software is the middleware RoSe. The local navigation was adapted to be able to operate on narrow forest tracks which are sometimes barely wider than the robot itself. The presented mapper allows navigating in areas with bad GPS reception, like forests.

REFERENCES


