Conclusions from an Object-Delivery Robotic Competition: SICK Robot Day 2014

Sebastian Buck, Richard Hanten, Goran Huskić, Gerald Rauscher,
Alina Kloss, Jan Leininger, Eugen Ruff, Felix Widmaier, Andreas Zell
Cognitive Systems
University of Tübingen

***Abstract*—Contrary to controlled experiments in a laboratory, robotics competitions pose a real challenge to an intelligent autonomous system. To allow a good performance, robotic systems have to be very robust to unforeseen circumstances in a highly constrained time frame. In this paper we present the design, implementation and performance of the robotic system we developed for participation in the SICK robot day 2014, with which we achieved very good results, placing second, with the same maximum score as the first. The task of this competition was to alternately fetch and deliver objects in a simplified warehouse scenario. Participating robots had to be able to carry out different tasks, such as detecting filling and delivery stations, receiving and analysing bar-code labelled objects as well as navigating in an arena among three other robots.**

I. Introduction

Participation in a competition poses multiple constraints on the design and implementation of a robotic system. Contrary to many other situations, deadlines cannot be extended and the system has to perform autonomously on the first try. Every part of the system has to perform at the same time and every part has to collaborate with every other reliably. Competitions are therefore a challenge that motivate a design that is simple and efficient, yet robust to influences that cannot be controlled or anticipated. Smaller robotics competitions are an exercise in team work and act as benchmarks, just as well as larger contests like RoboCup, the DARPA challenge and others [1].

The SICK Robot Day is a bi-annually hosted competition by the SICK AG, Waldkirch, Germany, a well known producer of sensor systems. In times past the objectives have varied notably, from perception and interaction, as described by Scherer et al. [2], Masselli et al. [3], Cigolini et al. [4] and Fejfar et al. [5], to navigational tasks as depicted by Fredriksson et al. [6]. Additional difficulty has always been caused by multiple competing robots performing simultaneously in an arena that is small enough to provoke robots encountering each other.

In this paper we describe our resulting system for the participation in the SICK Robot Day 2014, in which four robots were competing at a time to fetch and deliver objects in a scenario resembling the tasks in an automated warehouse. Besides a mechanism for collecting and delivering the objects, participating robots had to be able to detect filling- and delivery stations, navigate autonomously, and at the same time avoid collisions with other robots. The goal was to deliver the most objects, where each correctly delivered object was awarded one point and each erroneous delivery one penalty point.

This paper is structured as follows. Section II states the requirements posed by the rules of the competition. Section III illustrates the design of the resulting system and presents the different components. Section IV presents experimental results. Section V summarizes the conclusions. In section VI we discuss the results and the lessons we learned.

II. Requirements

The task at SICK Robot Day 2014 resembled that of a warehouse robot. There existed an approximately circular arena with about $12\mathrm{m}$ diameter in which four robots competed at the same time. With a time limit of 10 minutes, each robot had to alternately collect labelled objects at *filling stations* and transport them to *delivery stations* based on the object label.

The octagonal collection area, where the robots had to approach one of four filling stations, was located in the central part of the arena. Once a robot had reached the designated filling spot, it had to signal the human operator to provide an object to be delivered. These objects were known to be wooden cubes of side length $6\mathrm{cm}$, which were labelled on each side with the designated delivery station using a bar code imprint. On the outer boundary there were four delivery stations, one for each possible object label. Whereas the filling stations could be selected freely, every delivery station could only be used to hand off one specific object type. Therefore robot-robot interactions were inevitable and had to be accounted for.

Both filling and delivery stations comprised of a ring of diameter $30\mathrm{cm}$ at a height of around $50\mathrm{cm}$ and a bull's eye target sign for precise positioning (see Fig.2). Delivery stations were additionally marked with a large sign displaying one of the digits printed onto the wooden cubes. The robots therefore had to be able to detect both the bull's eye and the number signs. Once a robot had reached a station, it had to activate a green signal lamp to indicate its intention to collect or deliver an object. Afterwards the rules specified, that a human operator would insert or remove a cube within at most 10 seconds.

To successfully participate at the competition, a robot therefore had to fulfill the following requirements:

- Detection and localization of number signs displaying the digits 1 to 4
- Detection and localization of bull's eye target signs
- Receiving a cube and detecting its presence
- Reading the imprinted bar codes
- Localisation and navigation in a shared space with three other robots

III. Design and implementation

Here we present the final system design and implementation we used for participating in the SICK Robot Day 2014.

A. The platform

Fig.1 displays one of the two identical robotic systems we have constructed based on hardware we used at RoboCup [7] a few years back. These robots are based on an omnidirectional base with three omnidirectional wheels. The original use case demanded relatively large robots to be able to interact with soccer balls. For this competition we lowered the center of gravity as much as possible to allow faster movement speeds without increased risk of toppling in case of emergency braking. For processing we installed an Intel Core2Duo P8700 dual-core processor with 2.53GHz clock speed and 2GB of RAM just above the motors in an open space previously unused.

Fig. 1. Front view of one of our two robots with a hopper on top. Visible sensors: Allied Vision Marlin (center), PointGrey Firefly (top, inside the hopper), SICK LMS100 (bottom).

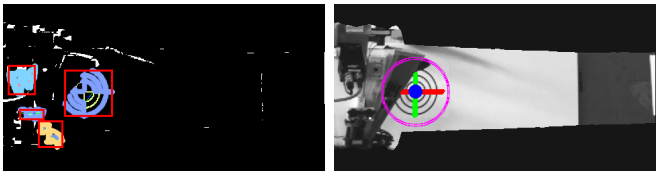
The platform was equipped with a hopper in the shape of a pyramid with a triangular basis standing on its tip, which we used to gather the cubes. Inside this hopper we mounted a PointGrey Firefly, which was used for detecting the cubes and reading their bar codes. Furthermore, we added a required green signalling lamp to indicate to human operators that the robot intends to receive or deliver a cube. We employed two laser scanners for obstacle avoidance and localization, a front facing SICK LMS100 and a rear facing SICK TiM551 which together allowed for 360° vision with only little blind spots to the sides of the robots. Additionally we used a front facing Allied Vision Marlin camera for sign and target detection. Front and rear are just defined for discussion, since the platforms are omnidirectional and therefore do not have a constrained movement direction.

Our robots were running the Ubuntu 14.04 operating system, on which we implemented all our software using ROS [8], OpenCV [9] and a self developed framework for prototyping and experimentation with cognitive systems called cs::APEX[1].

B. Target detection

(a) Number sign and partly occluded bull's eye target

(b) Masked image



(c) Regions of interest

(d) Resulting detection

Fig. 2. Bull's eye detection is performed on regions of interest. These are determined by first projecting the laser scan onto the image plane to mask everything but the border [2(b)] and then extracting [2(c)] and analysing [2(d)] dark patches with light backgrounds.

The primary part of localising filling and delivery stations was the detection of bull's eye patterns as seen in figure 2(a). These patterns were placed directly below the wooden rings through which the objects were thrown by the human operators and had to be used to accurately stop below.

Since the height of the surrounding border of the arena was known to be about 50cm, we first created a mask to quickly dismiss large parts of the camera image by extracting a band of 35cm around the expected height of target signs. This was achieved by projecting each point $p_i = (r_i \cos(\alpha_i), r_i \sin(\alpha_i))^T$ of the front laser scanner onto the camera image using the camera calibration matrix A and the static transform between camera and laser scanner frames ${}^C T_L$ to calculate the image coordinates

$$(u, v)_i = A \cdot {}^C T_L \cdot r_i \left(\cos(\alpha_i), \sin(\alpha_i), z\right)^T \qquad (1)$$

where z is not measured but specified and r_i and α_i are the range and angle of the ith ray, respectively.

The points $(u, v)_i$ were calculated twice, first for a height of $z = 0$ and then for $z = 35\text{cm}$, resulting in two sets L_{low} and L_{high} where L_{low} corresponds to the projection of the laser scan and L_{high} is shifted upwards. Finally, the polygon between L_{low} and L_{high} was filled to generate a matte as shown in figure 2(b). The masked image then was examined for

[1] http://www.ra.cs.uni-tuebingen.de/software/apex/

regions of interest by first applying a blackhat morphological operator and then performing a connected component analysis using cvBlob [10]. Large components were normalized to improve robustness against lighting changes and then further processed.

We calculated Canny edge features [11] and applied the probabilistic Hough transform [12] in order to extract the horizontal and vertical line in the bull's eye pattern. Detected lines were processed using various heuristics to reject short or skewed lines. Every pair of the remaining lines was tested for intersection. If the angle of intersection was near $90°$, a target message was published using the reprojection of this intersection as the position.

C. Sign detection

Delivery stations were marked with signs displaying the numbers 1 to 4, whereby the number represented the type of cube that could be delivered. The robot therefore had to be able to detect these signs and read the depicted number.

To solve this problem we compared two artificial neural nets, the one we used at SICK robot day 2010 based on the Stuttgart Neural Net Simulator (SNNS) which was described in detail in [2] and a later approach using JANNLab [13]. The need for an alternative classifier stemmed from the observation that some numbers could not be detected as well as others. The new neural net was implemented as a *multilayer perceptron* (MLP) network as well.

To optimize the runtime cost, we also used a laser projection mask as described in III-B. Here we extracted a band above the wall in which the signs had to be seen, which further reduced costs compared to the previous approach in [2].

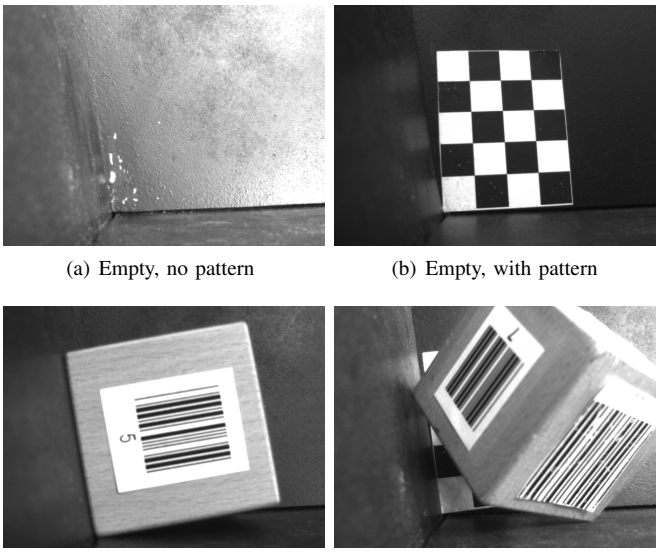
Detected signs were mapped in a sign map m_s for future reference, such that the robot did not have to search for the appropriate sign for every delivery run. We have implemented a Gaussian mixture model to be able to model hypotheses in a way robust to false detections. For the model of a sign we used a two dimensional Gaussian distribution for the x-y position and a von Mises distribution for the orientation. As shown by Bishop et al. [14], the maximum-likelihood solution $\bar{\theta}$ for the orientation of a hypothesis is then given by

$$\bar{\theta} = \operatorname{atan2}\left(\frac{1}{N}\sum_{i=1}^{N}\sin(\theta_i), \frac{1}{N}\sum_{i=1}^{N}\cos(\theta_i)\right), \qquad (2)$$

where θ_i are the individual measurements of the orientation and N is the number of measurements. For path finding we referred to this sign map and chose the target pose based on the best hypothesis.

D. Cube and bar code detection

After collecting a new cube, the target delivery station had to be determined. To get the numerical value of the cube's bar code we used the ZBar library [15]. Bad lighting conditions or a wedged cube could cause a failure of the bar code analysis. Therefore we needed a second robust system to frequently check whether an object is in the hopper or not.



(a) Empty, no pattern (b) Empty, with pattern

(c) Cube in optimal position (d) Cube wedged

Fig. 3. The cube hopper camera scene.

1) Intensity Standard Deviation: Our first approach to realize a cube detection was to calculate the standard deviation of the intensity values. The robot's hopper is painted black, so the standard deviation of the empty camera scene is minimal, as can be seen in Fig.3(a). The moment a cube enters the scene, the standard deviation rises considerably.

A problem with this approach is, that it heavily relies on a well configured camera which adapts to the lighting conditions. Once the shutter adoption fails, the approach does not work anymore. A change in lightness can be observed comparing Fig.3(a) and Fig.3(c).

2) Neural Network: As JANNLab was already a part of the software infrastructure, the idea was born to try out a neural network based approach. The basic concept was to train a neural network to detect the characteristics of a pattern independent of the lighting conditions and resulting image brightness.

A multilayer perceptron was applied to that task. The network was trained using data generated under several different lighting conditions. We also took data conditions in account which were more extreme than the competition conditions.

(a) Visible pattern (b) Occluded pattern

Fig. 4. Visualization of the intensity vector used for the MLP.

For training and usage of the network a vector of 100 intensity values was extracted from the pattern within the scene. The values were later on normalized to an interval from -1.0 to 1.0 to make them suitable for the neural network. In Fig.4 a visualization of a intensity vector using a Bézier color function is shown.

E. Localization and Mapping

To localize the robot in the arena, a graph based simultaneous localization and mapping (SLAM) algorithm was used. A plane localization algorithm, e.g. Monte Carlo localization was not suitable, because the exact map of the arena was unknown before the competition and the preparation time at the competition was too limited to build up a map. A SLAM algorithm using only scan matching, like the open-source module of the Hector framework by Kohlbrecher et al. [16], did not work, because the arena was too symmetric to detect correct movements from the scans. To our knowledge, the most accurate open-source implementation of a graph-based SLAM algorithm is Karto [17]. However, this algorithm assumes a static environment and is usually used for exploration and not for long term use. These characteristics cause two problems in our task.

There were other robots in the arena that moved and sometimes stood still. If another robot was not moving for a certain time it was included as occupied space in the map. This occupied space is remembered by the algorithms for several iterations, even if the other robot has moved away. This complicates path planning, because it adds unnecessary costs to the cost map. To solve this problem we preprocessed the laser scan with a jump distance segmentation filter. This filters a scan using the difference $\Delta r = |r_k - r_{k+1}|$ of two neighbouring rays r_k and r_{k+1}. If Δr exceeds a threshold T a new segment is started. A segment is removed if the euclidean distance of its start and endpoint is below a minimum value S_{min} or a above maximum value S_{max}.

The second problem is that Karto stores all matched scans. Every time a map is requested from the path planner, all n stored scans have to be integrated into a grid map. Therefore, the computational complexity and the time to provide a map increase linearly with the number of stored scans. Since long waiting times caused time-outs in the path planner, we had to limit the map creation time without decreasing the localization accuracy. This was done by limiting the number of scans being used for map update. To also include some older scans in the map, scan selection was divided in to three intervals. A dense interval used the scans s_k with index $n - 100 \leq k < n$, in the semi dense interval scans with index $n - 200 < k < n - 100$ and $k_{i-1} \leftarrow k_i - 10$ and in the sparse section the scans with $n - 1000 < k < n - 200$ and $k_{i-1} \leftarrow k_i - 100$ where used. In the background the whole graph with all scans is kept to achieve high localization accuracy from loop closures and to avoid drifting of the map.

F. Map analysis

One important reason for performing SLAM instead of driving reactively was to be able to use a map of the environment for determining the positions of filling and delivery stations. Delivery stations were assumed to be on the arena border. In case a station was unknown, the robot scanned the border systematically by driving clockwise around in the arena while looking in the tangential direction.

The filling stations were placed on every second side of an octagonal island at the center, whereby the side facing the starting position of the robot was guaranteed to be a station. To determine the exact positions of the four filling stations, we first classified every occupied cell in the grid map into *centre* and *border*. Hereby we assumed that the segmentation filter described in section III-E was able to completely remove every robot from the laser scans such that the map would only contain occupied cells for walls.

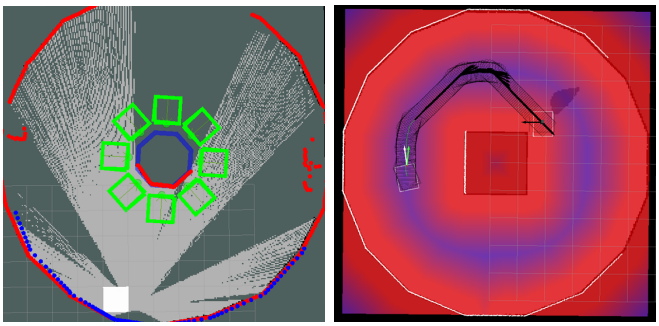
On all cells labelled as *centre* we performed RANSAC with an octagon model: We repeatedly sampled two points assuming they were the endpoints of one side s_1 of the octagon. We then extended s_1 in the direction of the centroid of the map to a full octagon by adding s_2 to s_8. Then we selected side $f_1 = s_i$ as a filling station side, for the i which minimized the angle between side s_i and the starting position. We then added every second side: $f_2 = s_{(i+2) \mod 8}$, $f_3 = s_{(i+4) \mod 8}$, $f_4 = s_{(i+6) \mod 8}$.

For $1 \leq i \leq 4$ we published the projection

$$p_i = c + \frac{f_i - c}{||f_i - c||} \cdot (||f_i - c|| + 0.3) \tag{3}$$

of the centre of side f_i as a filling station, where p_i was projected 30cm in front the wall, away from the centre c.

We then checked a rectangular box around p_i for laser scan hits in order to decide which filling station was free and which was in use. As shown in figure 5(a), we also performed this procedure for the remaining four sides, because we had assumed a square centre island and only had limited time to change the algorithm to detect an octagon.



(a) Analysed map showing the detected octagon and 8 free sides

(b) Cost map (simulation) used for path planning. High costs are red.

Fig. 5. RANSAC analysis of the grid map given by SLAM and generation of a cost map using the distance transform.

G. Navigation

We employed a full navigation stack including path planning and following. The map generated by the SLAM subsystem was preprocessed and then used for path finding. The resulting path was postprocessed and then used by a path following controller.

1) Map preprocessing: Before every request $r = (s, g)$ for a path from starting pose s to goal pose g, our navigation stack requested a grid map representation m of the current pose graph. m was then treated as an image and submitted to a morphological convolution to grow obstacles and close small unknown regions, whereas circular regions around s and g were kept unchanged to allow planning as closely to obstacles as possible. Afterwards, a combined map m' was computed, by firstly integrating the current readings from both laser scanners

into the map m. Using m' we computed a cost map m_c, by calculating the distance transform and then scaling and capping the values such that costs were maximal at obstacles and went to 0 for distances larger than $2.5\mathrm{m}$, which can be seen in Fig.5(b).

2) Path planning: Given the map m and a cost map m_c, we implemented a variant of A^* using the simple L_2 heuristic given by the norm $||\cdot||_2$. Since our robots were omnidirectional, we also implemented a simple omnidirectional motion model using the natural 8-neighbourhood of a grid map. For the past-cost we used $g'(q) = g(q) + m_c(q)$, where $g(q)$ denotes the known distance from s to the current configuration q and $m_c(q)$ the cost of configuration q. We manually tuned the scale of m_c such that resulting paths stayed as far away from obstacles as possible while not completely ignoring configurations q with $m_c(q) > 0$.

In a postprocessing step we used a heuristic to simplify path segments by recursively removing nodes in regions with no costs associated and checking if the resulting path was still valid. This was motivated by the fact that slanted paths in an 8-neighbourhood are often longer and more ragged than necessary due to only being able to represent multiples of $45°$ when expanding a node. After simplification we performed path smoothing and interpolation to allow for a better path following.

3) Path following: The path following controller is based on the idea presented by Mojaev et al. [18], which was further described and used in the dissertation of Li [19]. The idea is a simple control law based on an orthogonal projection of the vehicle to the desired path, as shown in Fig.6. The orthogonal projection is defined as an exponential function of the tangential component in the projected point on the path. If we denote the orthogonal projection as x_n, the tangential axis as x_t, and the initial distance from the path as x_{n_0}, we get the expression

$$x_n = x_{n_0} exp(-k x_t), \tag{4}$$

where k represents a positive constant which regulates the convergence speed of the orthogonal projection. The tangential angle of the exponential function is then

$$\phi_c = arctan(-k x_t). \tag{5}$$

If we consider the fact that the controlled robot is omnidirectional, its linear velocity in the world frame can be described as

$$\begin{aligned} v_x &= v_n cos(\alpha), \\ v_y &= v_n sin(\alpha), \end{aligned} \tag{6}$$

where $\alpha = \phi_c + \theta_t$. Here, v_n is the nominal velocity, θ_t the tangential angle of the desired path and α is the driving direction angle. The angular velocity is controlled independently using a PID controller. Stability analysis of the algorithm can be found in [19].

For the purpose of this competition, we have extended the algorithm by adding velocity control. The velocity was not constant, as in [18] and in [19], but it was rather depending on the curvature Maček et.al [20], distance to the obstacles, distance to the goal and angular velocity. The basic idea

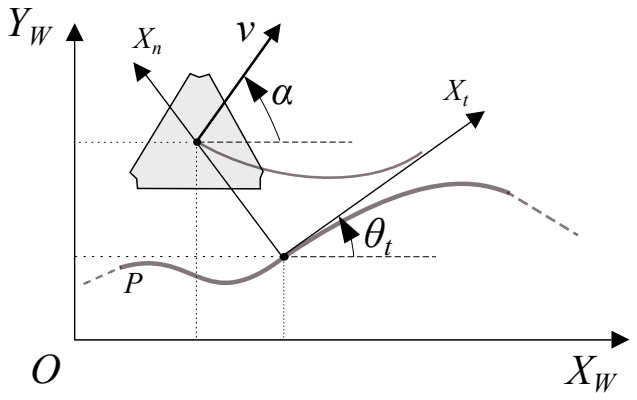


Fig. 6. The principle of the orthogonal projection path following algorithm.

from [20] was slightly changed, so that the velocity changes exponentially. The final expression for the velocity is then

$$v = v_n exp\left(-\left(K_\kappa \sum_{i=i_P}^{l} ||\kappa_i|| + K_\omega |\omega| + \frac{K_o}{d_o} + \frac{K_g}{d_g}\right)\right) \tag{7}$$

Here, $K_\kappa, K_\omega, K_o, K_g$ are positive constants used to adjust convergence, i_P is the index of the currently projected point on the path, and l is the index of the look-ahead distance. The curvature in a certain point is κ_i, ω is the angular velocity, d_o the distance to the nearest obstacle, and d_g the distance to the goal.

4) Obstacle avoidance: The path planner we used was fast enough to allow a simple obstacle avoidance scheme: Once we detected an obstacle that stayed on the current path for too long, the robot stopped and issued the system to generate a new path. We had developed several layers of safety: The path planner preferred paths that stay away from obstacles because of the cost map. The control in Equation 7 decreased the velocity of the robot in the vicinity of obstacles. We additionally employed a safety zone in the driving direction, such that if an obstacle was detected in this zone, the robot was stopped completely.

H. State machine

For high level planning we implemented the finite state machine depicted in figure 7. After waiting for a user to give the "go" signal by pressing a sequence of buttons on the remote control, the robot was completely autonomous.

A first state was introduced to explore the central area, such that our map analysis component was able to determine the position of the filling stations. This initial exploration consisted of driving a predetermined distance diagonally in order to allow the SLAM system to build up a map. Afterwards we iterated fetching and delivering cubes, where both states consisted of several sub states.

Fetching a cube, visualized in figure 8, included selecting a free filling station, planning and following a path there. Once the calculated target pose had been reached, the robot had to use the bull's eye target detection to orientate itself towards the station. It then drove forward until the distance to the wall was below $5\mathrm{cm}$, signalled the human operators and then backed

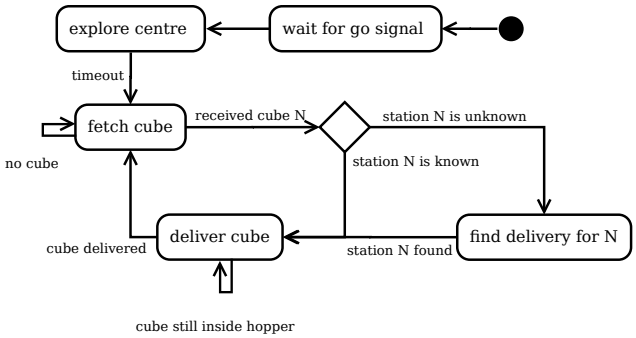


Fig. 7. The finite state machine we implemented. Error states were omitted.

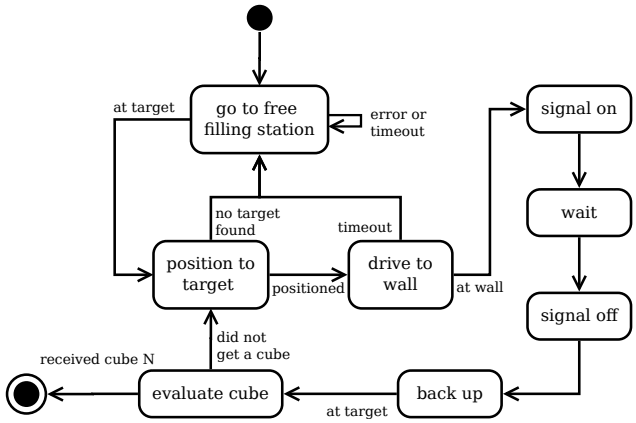


Fig. 8. The meta state *fetch cube* in figure 7.

up again. If at this point no cube had been received, e.g. if the target position had not been accurately reached, the robot issued a retry. Otherwise the cube was analysed and the sub state was left.

Delivering a cube was essentially the same as fetching a cube, only instead of going to a free filling station, the path finding module was requested to plan a path to the appropriate target station if it was known. Otherwise we switched to finding this unknown station by driving around the arena in a clockwise direction.

IV. Experimental results in the lab

Experiments have been performed in our laboratory and in a gym of the University of Tübingen. Some of the experiments have been repeated after the competition for more precise results in this paper.

A. Target detection

The performance of the bull's eye target sign detection was acceptable for the competition and worked in every instance. There is an inherent downside to our line extraction approach due to motion blur. Even at relatively low rotational speeds, motion blur was too much of an influence, for target signs to be detected at larger distances. However, we only needed to detect these signs in the final step of approaching a station. The map analysis was sufficient to allow planning a path that ended directly in front of a filling station, where the signs were clearly visible. In the case of delivery stations, we relied on detecting number signs at larger distances and then used path finding to position the robot in front of these number signs, where again the target signs were easily visible. That is why the targets only had to be robustly seen at distances of $2\mathrm{m}$ or less, in which case motion blur didn't affect the detection rates too much.

B. Sign detection

For evaluation of the sign detection we refer to Scherer et al. [2]. The approach has not been modified, except for masking out irrelevant parts of the camera image, which had an impact on the runtime but not on the classification performance. The new approach based on JANNLab looked very promising and achieved higher detection rates in our test environments. On site at the competition, however, there were a few reproducible false positive detections caused by the environment. This is why we decided to use the old neural net, accepting the lower true positive rate for the benefit of far fewer false positives. Due to the lower true positive rate, we had to reduce the robots top speed to ensure that signs could still be mapped accurately while exploring the environment.

C. Cube and bar code detection

The standard deviation and MLP based detections were evaluated on datasets containing images taken under several different poses and differing lighting conditions. We considered that either an object is already in the hopper or the hopper is completely empty. Also we present the raw results of the two approaches without any filter within the processing pipeline.

In table I the results of the evaluation can be observed. The labels in the confusion matrices indicate the state of the hopper. It is either *empty* or *full*.

The standard deviation based approach was tested on 1306 frames without the background pattern used by the second approach. The variance threshold was set to 50 to detect an object within the camera scene. The MLP based approach was tested on 1325 different frames containing the pattern.

Std Dev.

		Prediction	
		empty	*full*
Actual	*empty*	563	150
	full	158	435

MLP

		Prediction	
		empty	*full*
Actual	*empty*	662	14
	full	1	648

TABLE I. Confusion matrices for cube detection approaches.

The results lead to the conclusion, that the MLP based approach is much more robust than the simpler approach. There is almost no logical filtering afterwards necessary. In the end the standard deviation based approach suffers from being highly dependent on the behaviour of the camera.

D. Localization

The segmentation filter produced good results because the maximum robot size was limited to $0.6\mathrm{m}$ by the rules. We set the segment size minimum value to $S_{min} = 1.2\mathrm{m}$ to be sure that a segment is removed even if two robots are standing close to each other. The threshold for the minimum distance of rays was set to $T = 0.1\mathrm{m}$ so that small robots could still be separated from a wall, and the octagon was still seen as one

big segment. A comparison of a map created with and without the scan preprocessing can be seen in Fig.9.

Our method of limiting the map creation time also showed good results as we could limit it to 62ms, while at the standard algorithm the time would increase with 0.5ms per scan.

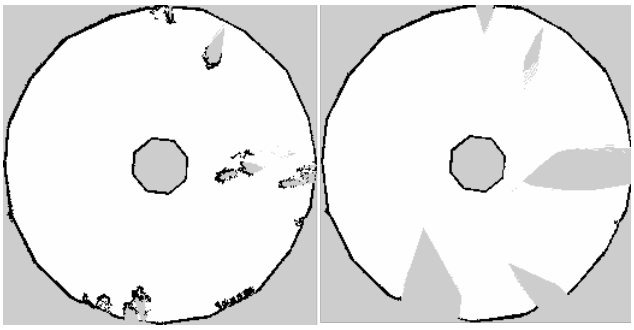


Fig. 9. Grid map without preprocessed scans (left) and with segmetation filter preprocessed scans (right)

E. Navigation

1) Path planning: Our path planner was tuned to a map resolution of 5cm. The worst case for running time for path finding is a goal pose that cannot be reached, because then the search exhausts the complete state space. We tested this by using a map of the arena and finding a path to a point on the outside of the map, which on average took around 120ms. Additional costs came from post processing of a planned path, which took on average about 20ms. This short response time allowed us to use the obstacle avoidance scheme we described in III-G.

2) Path following: As already mentioned, the original path following algorithm used a constant velocity, controlling only the direction angle. This was tested on our robot in a test arena and it worked well for the velocities up to 1m/s. However, there was a problem of instability at higher speeds. Since the velocity was constant, the robot would drive with full speed in a curve, or near the obstacles, which would sometimes lead to deviating too much from the path, or hitting the obstacles. Also, it would drive with full speed until it would reach its goal. In that moment, it would try to brake instantly, which would always lead to behaviour close to roll over.

The solution to this problem was to extend the original algorithm, as already described in the previous section. Our approach gave us satisfying results up to a top speed of 2m/s, since the robot would slow down in critical situations, so that the following was more accurate, and the behaviour more stable and safer. In the competition, the nominal speed was 1.5m/s, just for safety reasons.

The algorithm was firstly developed in MATLAB. Gazebo was used for further simulation tests before implementing it on the robot. In Fig.10 a sample desired path (green) can be seen, as well as the real path performed by the robot (blue). In this example, the robot has reached its goal position, and it was oriented towards a look-at point (red dot) all the time. The orientation of the robot is represented by a white arrow marker.

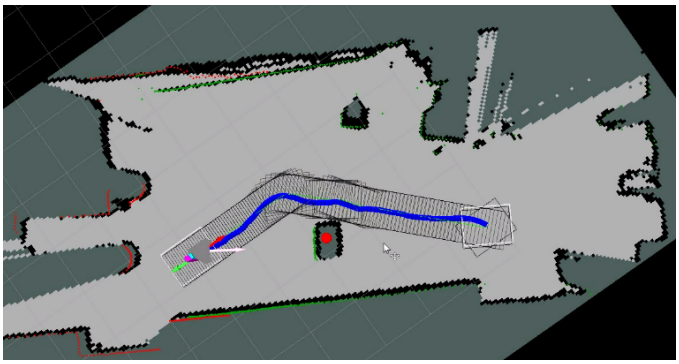


Fig. 10. Desired vs. driven path.

V. Results in the Competition

The competition took place on October 11, 2014 in Waldkirch, Germany. Every attending team had a few hours before the competition to test their systems on site. The arena, which nobody had seen before, was mostly as expected. There were only four delivery stations as opposed to the expected worst case of ten stations. The size of the arena matched our expectation, yet the central area for filling stations was not as we had anticipated: The rules had specified that the filling stations would be located at the 4 sides of a square island in the centre of the arena, which ended up being an octagon with 4 unused sides as described in Section III.

Every robot had two runs of 10 minutes each, whereby the better run was to be used for ranking. Out of the 14 competing teams, five managed to score one or more points. Our robot managed to correctly deliver six cubes in ten minutes, resulting in a second place with as many points as the winner RIMLab with their robot PARMA[2]. The tie was broken by consulting the other run, in which our robot could only deliver three cubes, because of a malfunction which resulted in our robot standing still for five minutes after it had already moved for about a minute.

The delay was most likely caused by a driver problem we had infrequently observed a few times earlier, in which the driver for our main laser scanner would freeze up and not publish any new data. This is in accord with the log files, which show that the laser scanner reported obstacles directly in front of the robot when clearly there were none. Since this problem had been observed earlier, we had implemented a watch dog process to restart everything related to the laser scanner subsystem in cases when no data was transmitted for a while, but it seems that multiple restarts were required.

There are a few optimization possibilities which would have allowed us to decrease the time per object in order to deliver at least one more object:

- In contrast to many other teams, our robot waited for the full 10 seconds on both filling and deliver stations for safety reasons. For six delivered objects, this alone resulted in 2 minutes of waiting time, which mostly could have been avoided.

[2]http://rimlab.ce.unipr.it/SickRobotDay2014.html

- To avoid CPU overload, which would have been detrimental to our SLAM module, we decided to disable parts of the system when they were not needed, and reduce the frame rates of our cameras. This resulted in the need to wait in front of expected bull's eye signs to increase the confidence in the observation, which costed another unnecessary 5 seconds per station and about one minute overall.
- We used a maximum velocity of $1\mathrm{m/s}$ in the first run, which was already relatively fast compared to other teams but was only a safety precaution to absolutely avoid any dangerous situation.

For the second run we were able to quickly change our state machine and decrease waiting times to the absolute minimum. We also further increased the driving speed to $1.5\mathrm{m/s}$, however the effort was wasted because of the above mentioned inactivity problem.

VI. Discussion

During the preparations and the competition itself we have learned valuable lessons concerning the implementation of autonomous robotic systems under time constraints that we want to discuss and share.

A. Encapsulation of assumptions

It has proven valuable to formulate all the assumptions about the target environment and model them explicitly in the resulting software. For example, one assumption was "the centre area is square", which we modelled as a class `Square`. Once we realised that the assumption was not fulfilled, we only had to implement another model `Octagon`. Unfortunately our model was not general enough, because the assumption "every side of the centre contains a filling station" was not explicitly modelled and had to be implemented spontaneously, which resulted in a suboptimal system.

Another assumption was "all numbers between 0 and 9 may or may not exist as codes". This not only affects the false positive rejection in bar code reading and number sign detection, but also higher level functions, for example in a guessing scheme in case a bar code could not be read. Other examples we modelled are the maximum size of other vehicles, the geometry of stations and that the arena was symmetric.

B. Rapid prototyping in a collaborative environment

One of the most important themes in software development is modularization, especially with multiple collaborating developers. Using ROS for robot specific parts and cs::APEX for cognitive algorithms, we relied on pure message passing interfaces, allowing us to quickly and efficiently develop and reuse parts of the system with little overhead. Modularity also allowed us to simply switch out the different implementations of number sign detectors and map analysis components under strict time constraints.

Acknowledgement

The authors would like to thank Sebastian Otte for providing his implementation of different artificial neural network algorithms. We also want to thank the SICK AG for sponsoring each participating team with a SICK TiM551 laser scanner.

References

[1] S. Behnke, "Robot competitions-ideal benchmarks for robotics research," in *Proc. of IROS-2006 Workshop on Benchmarks in Robotics Research*, 2006.

[2] S. A. Scherer, D. Dube, P. Komma, A. Masselli, and A. Zell, "Robust Real-Time Number Sign Detection on a Mobile Outdoor Robot," in *Proceedings of the 6th European Conference on Mobile Robots (ECMR 2011)*, Örebro, Sweden, Sep. 2011. [Online]. Available: http://www.cogsys.cs.uni-tuebingen.de/publikationen/2011/scherer11.pdf

[3] A. Masselli, R. Hanten, and A. Zell, "Robust real-time detection of multiple balls on a mobile robot," in *Mobile Robots (ECMR), 2013 European Conference on*. IEEE, 2013, pp. 355–360.

[4] M. Cigolini, A. Costalunga, F. Parisi, M. Patander, I. Salsi, A. Signifredi, D. Valeriani, D. L. Rizzini, and S. Caselli, "Lessonslearnedinaballfetch-and-carryroboticcom etition," *Journal of Automation, Mobile Robotics & Intelligent Systems*, vol. 8, no. 1, 2014.

[5] P. Fejfar and D. Obdržálek, "Smart and easy object tracking," *Vera Kurková, Lukáš Bajer (Eds.)*, p. 28.

[6] S. R. H. Fredriksson, D. Rosendahl, and K. H. A. Wernersson, "An autonomous vehicle for a robotday."

[7] K. Kanjanawanishkul and A. Zell, "Path following for an omnidirectional mobile robot based on model predictive control," in *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE, 2009, pp. 3341–3346.

[8] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[9] G. Bradski, *Dr. Dobb's Journal of Software Tools*, 2000.

[10] C. C. Linán, "cvBlob," 5 2014. [Online]. Available: http://cvblob.googlecode.com

[11] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 6, pp. 679–698, Jun. 1986. [Online]. Available: http://dx.doi.org/10.1109/TPAMI.1986.4767851

[12] J. Matas, C. Galambos, and J. Kittler, "Robust detection of lines using the progressive probabilistic hough transform," *Computer Vision and Image Understanding*, vol. 78, no. 1, pp. 119–137, 2000.

[13] S. Otte, D. Krechel, and M. Liwicki, "Jannlab neural network framework for java." in *MLDM Posters*, P. Perner, Ed. IBaI Publishing, 2013, pp. 39–46. [Online]. Available: http://dblp.uni-trier.de/db/conf/mldm/mldm2013p.html#OtteKL13b

[14] C. M. Bishop *et al.*, *Pattern recognition and machine learning*. springer New York, 2006, vol. 1.

[15] J. Brown, "Zbar bar code reader," Februrary 2013. [Online]. Available: http://zbar.sourceforge.net

[16] S. Kohlbrecher, J. Meyer, T. Graber, K. Petersen, U. Klingauf, and O. von Stryk, "Hector open source modules for autonomous mapping and navigation with rescue robots," in *RoboCup 2013: Robot World Cup XVII*. Springer, 2014, pp. 624–631.

[17] R. Vincent, B. Limketkai, and M. Eriksen, "Comparison of indoor robot localization techniques in the absence of gps," *Detection and Sensing of Mines, Explosive Objects, and Obscured Targets XV*, vol. 76641, pp. 76 641Z–76 641Z–5, apr 2010.

[18] A. Mojaev and A. Zell, "Tracking control and adaptive local navigation for nonholonomic mobile robot," in *Intelligent Autonomous Systems (IAS-8)*. Amsterdam, Netherlands: IOS Press, Mar. 2004, pp. 521–528.

[19] X. Li, "Dribbling control of an omnidirectional soccer robot," Ph.D. dissertation, Cognitive Science Department, University of Tuebingen, 2009.

[20] K. Maček, I. Petrović, and R. Siegwart, "A control method for stable and smooth path following of mobile robots," in *Proceedings of the European Conference on Mobile Robots*, 2005.