Universiteit
van
Amsterdam

IAS technical report IAS-UVA-12-02

UvA Rescue Technical Report: A description of the methods and algorithms implemented in the UvA Rescue code release

Arnoud Visser
Intelligent Systems Laboratory Amsterdam,
Universiteit van Amsterdam
The Netherlands

Abtract: This technical report gives the background documentation behind the competition code of the UvA Rescue Team, who participates in the RoboCup Simulation League. The described code is used in the Virtual Robot competition, where a team of robots, guided by a single operator, has to find as many victims as possible in a devastated area.

Keywords: Robotic Architecture, Multi-Robot Exploration, Behavior Based Control, Perception algorithms, Simultaneous Localization and Mapping

IAS
intelligent autonomous systems

Contents

Intelligent Autonomous Systems
Informatics Institute, Faculty of Science
University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam
The Netherlands
Tel (fax): +31 20 525 7461 (7490)
http://www.science.uva.nl/research/isla/

Corresponding author:
A. Visser
tel: +31 20 525 7532
A.Visser@uva.nl
www.science.uva.nl/~arnoud

1 Introduction

1.1 About the team

The UvA Rescue Team has a long history. The first participation in the Rescue Simulation League was by Stef Post and Maurits Fassaert, who competed in the 2003 competition in Paduva [45, 44]. In 2006 the first Virtual Robot competition was held. Max Pfingsthorn and Bayu Slamet participated in this competition and won the Best Mapping award [39, 51]. The team from Amsterdam started a cooperation with Oxford University in 2008, which continued for 4 years [20]. In 2012 the team operated again under its original name; the UvA Rescue Team.

During those years the team won several prices[1], published several journal articles [5, 3, 12], book chapters [43, 40, 69, 50, 2, 37, 17], conference articles [42, 14, 63, 36, 65, 67, 66, 21, 47, 61, 30, 6, 4, 68, 22, 32, 16, 56, 25, 23, 24, 60, 71, 35, 74, 34] and theses [1, 44, 51, 13, 49, 46, 31, 70, 52, 15, 27, 7, 38, 20, 73, 55, 33, 10, 9, 29, 48]. It described their approach every year in a Team Description Paper [39, 65, 64, 58, 62, 11, 59] and published their source code[2] with a public license.

1.2 About the document

The intention of this document is to give an overview of the code produced by the UvA Rescue team. This overview could be of interest for teams that like to enter the RoboCup Rescue Simulation League Virtual Robot competition, yet this document is mainly written to give new team-members of UvA Rescue team a head start.

Chapter 2 starts with a short introduction how to install the software, including the dependencies on other softer. Chapter 3 gives an introduction to the architecture behind the software. Chapter 4 gives an overview of how sensor data is processed. The sensor data of the different robots is registered on a global map, which is described in Chapter 5. Chapter 6 gives an overview of the implemented behaviors. Chapter 7 is the conclusion.

1.3 Changes in the code

The original code for the first Virtual Robot competition in 2006 was written in C++. The agents were completely autonomous; the code did not have much of a user interface. A user interface is quite easy to make with the .Net framework, so the same algorithms were reimplemented in Visual Basic. At a first glance C# would have been a more logical choice, but the developer (Bayu Slamet) had more experience with Visual Basic. The power of Visual Basic was demonstrated when Tijn Smits tried to implement the connection to the image server in C++. Both the support for sockets and streams is rudimentary in C++. After two weeks struggling Tijn in C++ Tijn implemented the same code in half a day in Visual Basic and wrote in his log[3]:

> *"I discovered that coding in VB saves a lot of time as it is less complicated and corrects syntax and cross-references code on the fly."*

[1] `http://www.jointrescueforces.eu/wiki/tiki-index.php?page=Achievements`
[2] `http://www.jointrescueforces.eu/wiki/tiki-index.php?page=Downloads`
[3] `http://student.science.uva.nl/~tschmits/log.html`, May 17, 2007.

The code is maintained on a svn-repository, which allows to maintain a record of the changes made in the code. The messages with each commit are stored in the distributions in the files `revisions2006.txt` ... `revisions2012.txt`. This files can also be generated by the calling script `Tools/add_revisions.sh` in a Cygwin environment (with the right year commented out). This is a small overview of what happened during the years:

2006 Revision 1:251
Both Bayu and Max made only sparsely use of the comment field of a commit. Actually, the developments of the C++-branch is better documented after the competition [51]. The developments on the C++ branch actually continued until May 2007 (rev. 484), with contributions from e.g. Matthijs Spaan, Xingrui-Ji, Luis Gonzalez and Laurentiu Stancu [54].

2007 Revision 252:671
In 2007 Bayu was implementing different ScanMatching algorithms to further improve the SLAM algorithm [40], while Tijn Smits was working on the omnidirectional camera [49]. In the meantime the protocol for relaying the DRIVE commands and SEN messages was made (and tested in the semi-finals).

2008 Revision 672:1349
In 2008 Bayu reimplement the ScanMatching with QuadTrees [68]. In the meantime also the frontier exploration [66, 67] was implemented as part of the Visual Basic code. Already at the German Open 2008 the control to the AirRobot (GPS based) was introduced.

2009 Revision 1350:1913
In 2009 Julian de Hoog added support for the Kenaf robot and Helen Flynn implemented automatic victim detection [16]. In the experimental branch Steven Roebert [47] and Gideon Maillette de Buy Wenniger [30] were able to interpret the omnidirectional images for navigation.

2010 Revision 1914:2163
In 2010 Okke Formsma implemented way-point following into the behaviors.

2011 Revision 2164:2228
In 2011 Nick Dijkshoorn made the code much more stable and mature.

2012 Revision 2229:2271
In 2012 the code was made much faster. At faster speeds a memory leak became visible, which was partly solved.

One of the main issues in the code was transmitting images over the wireless link. For a long time images were not correctly requested, which meant that competition were driven purely on the map. When the images were finally correctly transmitted, problems with synchronizing the stream with laser-scans and images became an issue (which seemed even more difficult on multi-core machines).

2 Getting Started

The UvA Rescue Team code controls a team of robots spawned into simulation. The simulation environment is USARSim, which is an environment based on the Unreal Engine. Three different versions of USARSim are available, one based on UT2004, one on UT3 and one on UDK. Those different versions are available on respectively cvs-, svn- and git-rtepositories. The UT2004

version contains most robot and sensor models and those models contain validated error models. Yet, UT2004 can no longer be bought and this version is no longer maintained. UT3 was an attempt to reproduce the same functionality on a different game engine. Unfortunatelly, the collision model never worked well, so only competitions on a flat floor could be held. This issue was solved for UDK, what is also a version which is freely available and much better documented. Yet, this development is still quite recent and the models didn't reach the richness and completeness of UT2004 (yet).

The simulation environment USARSim is a prerequisite. Without this environment only experiments with logfiles can be performed. Instructions how to download and install USARSim can be found at sourceforge[4].

2.1 Getting the code

The code is available from the Joint Rescue site[5], including instructions how to build an executable from this code. For UvA Rescue team it is easier to directly install the code from the svn-repository[6], because the download consists of the different contributions of each year which have to be installed over each other.

2.2 Building the code

To build the code of the UvA Rescue Team one need Visual Studio with the language packages C# and Visual Basic. Central in Visual Studio is the solution-file, which is a container for several projects. Each project has to be of the same language, but the solution can combine projects from different languages. Note that the combination of a C++-project with C#- or Visual Basic code is not trivial, because the latter two produce managed code, while C++ normally produces unmanaged code. The benefit of managed code is the compiled code is stored together with all metadata that describes the classes, methods, and attributes of the code you've created. Note that Visual Studio is able to produce managed code from C++, by linking the code to the Common Language Runtime libraries, instead of the native Runtime libraries[7].

The instructions to build the code are given in the `readme.txt` which accompanies the code and are actually quite simple:

- Open UvArescue2012/UvARescue2005.sln or UvArescue2012/UvARescue2010.sln (depending on your version of Visual Studio)
- Build UsarClient
- Build UsarCommander

The result is two executables: `UsarClient.exe` and `UsarCommander.exe`. In the Configuration Manager the dependencies of both executables are specified. If configured correctly, those two build commands also result in the build of all underlying libraries. If one of the libraries fails to build, try to build it separately. Sometimes there are dependencies between libraries. When the libraries needed for `UsarClient.exe` and `UsarCommander.exe` are built for the first time in a wrong order, the initial build can fail. After building the libraries separately, the overall build of `UsarClient.exe` and `UsarCommander.exe` should be no longer a problem.

[4] http://sourceforge.net/apps/mediawiki/usarsim/index.php?title=Installation

[5] `http://www.jointrescueforces.eu/wiki/tiki-index.php?page=Downloads`

[6] `svn://u013154.science.uva.nl/Roboresc/2011/competition`

[7] When you create a new C++-project, one can choose between several templates (AFC, CLR, General, MFC, Test and Win32). The choice between managed code and unmanaged code (native) is between CLR template and a Win32 template (AFC and MFC templates prepare the application for use of COM services). If you have a existing C++-project, it is an option in the Configuration Properties at the tab 'Project Defaults'

2.3 Starting the code

The `readme.txt` also gives instructions how to start the application:

- Make UsarCommander the default executable by making it the StartUp-project (right click on the project in the Solution Explorer-window).
- Configure a robot team consisting of a ComStation and a number of robots (i.e. P3AT). This can be done by adding a number of robots (by using the +-button), configure each robot (by clicking on the configuration-button (gear icon) and loading a configuration file with a name which corresponds to the USARSimRunMap you like to explore), and configure the networks settings of the team (button with world-icon).
- Only for the ComStation the radio button 'Spawn for Commander' is active.
- Start a Run (green arrow).
- Start USARSim by executing one of the scripts in the directory `./USARSimRunMaps`. Also start the Wireless Simulation Server (which can be found in `./USARSimTools`).
- Spawn the ComStation.
- Spawn for each robot the Proxy at the current machine and an UsarClient at another machine.
- An UsarClient is started from the commandline with the command `'UsarClient.exe -n <name> -ac <agentconfigfile> -tc <teamconfigfile>'`.
- Use the controlbuttons of each Proxy to direct the robots through the environment. The shared map is constantly updated when the robots are driving around.
- Have fun!

The result of all programs started should initiate the connections as displayed in Fig. 1:

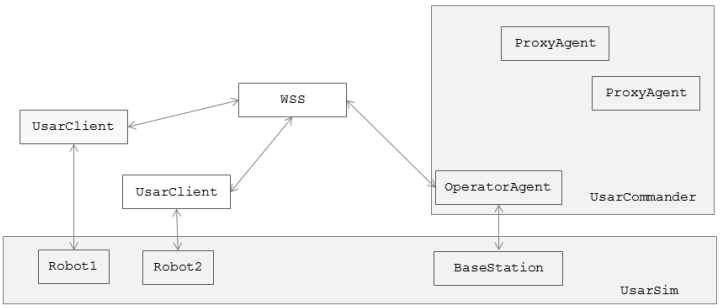


Figure 1: The connections between the programs in a competition setting.

For the operator, the programs running on his computer will a Graphical User Interface (GUI) and several consoles, as is the screenshot displayed in Fig. 2.

During development, one can skip the Wireless Simulation Server by activating also for the robots the 'Spawn for Commander' radio-button. In that case no UsarClient.exe commands have to be given. Instead of ProxyAgents a number of BehaviorAgents will run as a thread inside UsarCommander, which will make direct connection to `USARSim` and spawn `Robot1` and `Robot2`. Directly connected to USARSim the code is faster and easier to debug. Yet, remember to test your algorithm also in a competition setting (with the Wireless Simulation Server and several UsarClients.

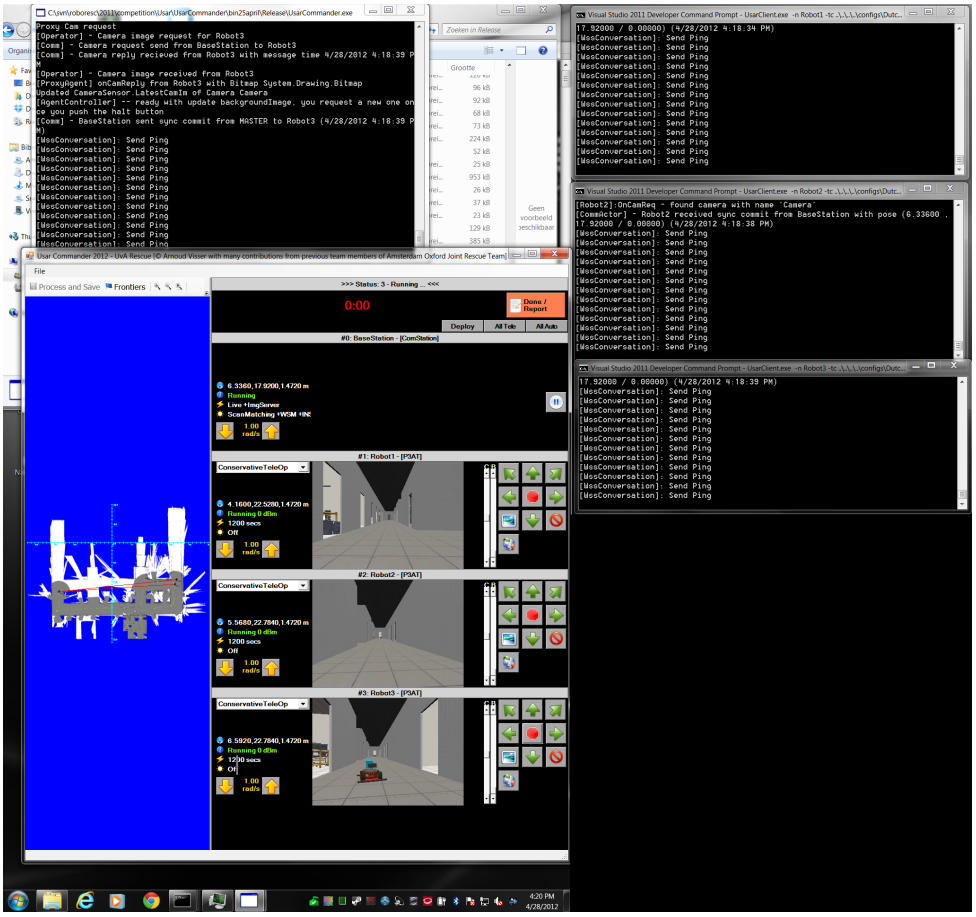


Figure 2: Screenshot of GUI and robot consoles of the UvA Rescue Team during a Virtual Robot competition (Dutch Open semi final 2012).

3 Architecture

As described in previous section, the UvA Rescue Team code consists of two programs: `UsarCommander` and `UsarClient`. Both programs consist of a project with a single file, the remainder of the functionality is available in libraries both programs share. The following libraries can be distinguished:

- **UsarLib:** this library consists of three folders. The *Team*-folder defines the different types of agents which can be started. Examples are UsarSlamAgent (which makes a map) and UsarSkinDetAgent (which uses image processing to detect victims). The classes in the Team-folder are used by both `UsarCommander` and `UsarClient`. The other two folders consist of GUI-functionalities which is only used by `UsarCommander`. The *Dialogs*-folder has the dialogs, which are mainly used before and after the competition run. The *Views*-folder is used during a competition-run, and displays on the left the map (with the position of multiple robots) and on the right the control windows for each robot which also displays the sensor updates for each robot.

- **Agent:** this library consists of seven folders. The *Actors*-folder contains the functionality to issue control-commands to the robot: the actuation. The *Agents*-folder is the container class for the robot: for instance it is possible to mount a number of sensors and actors to the robot (dependent on the type of robot). Part of the agent is the Worldview, this is the

local copy of the manifold containing the information that has reached this agent. The *Behavior*-folder contains behaviors and motions. Motions are a sort of state-machines which rules how to react on certain sensor-events. Behaviors are sequential and/or concurrent combinations of motions. The *Config*-folder contains the logic to synchronize the settings as used in the program with a configuration file. The dialog to change the configuration is part of the UsarLib library. Configuration files can also be changed by hand: the human readable format is used.

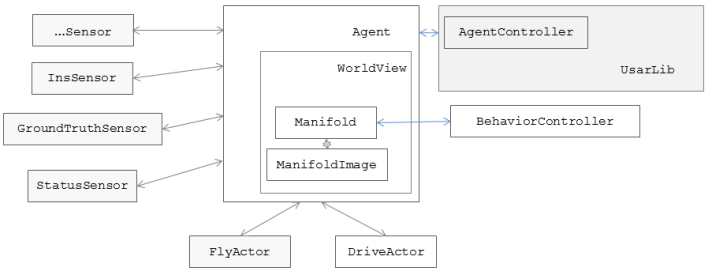


Figure 3: The Agent mounts a number of Actors and Sensors (gray errors) and publishes changes to a number of observers (blue errors).

The *Driver*-folder contains two important types of drivers: the LiveDriver and the LogDriver. The LiveDriver contains the interface to the simulator. It runs a separate thread which maintains the queues of commands to be send and sensor messages to be read. The LogDriver reads a logfile and issues the information in the same way as the LiveDriver. Notice that there exist logs which are a single file with a mixture of messages from several sources and there exists logs which are multiple files (in the same directory) each containing the message from a single source (sensor). In the latter case the messages from the different sources have to be synchronized on the basis of the timestamps provided for each message. The *Map*-folder contains five subfolders. The *Frontiers*-folder contains the functionality to extract frontiers from the map; the locations where the robot has to continue with exploration. The *Momento*-folder defines the summaries of the map-information as send over the Wireless Simulation Interface. An agent stores more information that it has perceived itself than it shares and receives with the other agents (to reduce the amount of information which has to go over the communication link). This means that the ProxyAgent also has a summary of the information that its alter ego in the field has. Each agent can have a number of observers. This is based on the publish-subscribe method. In the *Observations*-folder this functionality is implemented. Examples of observers are the behaviors (which need sensor-events) and the *Views* of the GUI. Next is the *Sensors*-folder. There are two types of sensors: SingleState- and MultiStateSensors. MultiStateSensors contain a queue of the unprocessed data. The Camera- and LaserRangeSensor are both MultiStateSensors.

- **Communication:** this library consists of two folders. The *Device*-folder contains three related classes. The WssDevice starts a WssListener and a number of WssConversations. Both the WssDevice, the WssListener and each WssConversation is a separate thread. The Communication library makes use of the System.Net.Sockets library and the UvArescue Tools.Networking library. The *Messages*-folder contains the messages which are exchanged between the Agents.

- **ImageAnalysis:** contains the algorithms to detect skin. Initially, the skin detection was based on color histograms[65]. Aksel Ethembabaoglu [13] used the same histogram

approach to follow a red robot. Later, Helen Flynn extended this with an algorithm based on shape[16]. Helen's code was based on OpenCV, a connection which worked but wasn't tested enough for competition usage. Another great work image processing work was performed by Steven Roebert and Gideon Emile Maillette de Buy Wenniger, which use an interface to Matlab to perform the image processing. Notice that this work was performed in another branch of the code (2008/sroebert) instead of the competition-branch. Part of their work (and of Tijn Smits) is still visible in the competition code, because when the camera is configured to look straight-up, it is assumed that this is omni-directional camera (which could be converted to a bird-eye view image [47]).

- **Math:** contains the vector and matrix classes. For several years the class Pose2D (x, y, θ) used in the exchange of information between the robots, but since the Dutch Open 2012 (with an elevated terrain) the class Pose3D is used $(x, y, z, pitch, yaw, roll)$. Many robotic applications (for instance ROS and LCM [57]) use a quaternion to represent their orientation without singularities (see also [33], section 3.4). In the UvA Rescue code the orientation is (still) represented with the class Vector3; the class Vector4 is used for homogeneous transformations. Notice that a generic Vector and Matrix class is also available inside the Third Party library Iridium.

- **SLAM:** this library consists of two folders. The *ScanMatcher*-folder contains several ScanMatching algorithms. Iterative Closest Point (ICP) is the classic algorithm, which is used as basis for more advanced algorithms. The ScanMatching algorithm were made far more efficient once they were implemented with quadtrees [68]. For each algorithm two variants are available (for instance WeightedScanMatcher and QuadWeightedScanMatcher). Actually, both are matched with quadtrees; the difference is that the quad-variants are matched against the whole (global) map, while the normal variants are matched against a local map (generated by combining a number of recent scans). The *ScanMatcher*-folder actually contains a single algorithm, ManifoldSlam, extensively described in [51].

- **Tools:** this library consists of nine folders. The *ArgParser*-folder contains the code to parse the arguments from the commandline. The *Config*-folder contains the code to store settings in a file. *GPX*-folder contains the code to store path in an xml-type of file, which can be read by geometric information systems. The GPX-format is one of the ogr formats. It was been used in 2009 when the path to victims had to be generated. The *Graph*-class is the base class of the Manifold-class, so this implementation plays an central place in the UvA Rescue Team code. *MapInfo*-folder contains the code to store path in another ogr format with extention mif. It is used to store the paths of the robots. *Networking*-folder extends the System's TcpClient and TcpListener class with a TcpConnection class. This code is used to build up the connection to USARSim and WSS. The *QuadTree*-folder contains the code to save points dynamically on a grid. The code is not only efficient with memory, but also allows searching very fast for the nearby points. This code is heavily used in the scanmatching. The *Threading*-folder implements the Regular- and PausableThread. The RegularThread is for instance used in the WssConvesation, WssDevice and WssListener. The PausableThread is used for the LiveDriver.

- **Third Party:** this library is a container for several external C#-libraries. *Iridium* and *Neodym* are part of the Math.NET Project[8]. Iridium is now discontinued and replaced by the Math.NET Numerics project. *LightFX* is an example of a wrapper class around a library written in C++. In this case the GamingSDK.dll which allows to control the colored

[8] http://mathnet.opensourcedotnet.info

leds in Dell XPS machines. For other systems this code is commented out. *SharpZLib* is library to compress data (for instance the raw images).

4 Perception

The perception algorithms can be distinquished in two branches. Leading in one branch is the LaserRangeData. When this data arrives, a new pose estimate is made. The data points are matched against previous point-clouds, as described in the next section. This search can start at the last known position, or can start at a location reported by another sensor (for instance the encoders, GPS or INS). ScanMatching algorithms are more efficient and more robust when correctly initialized, so this seed position is very important. Not all sensors can provide a complete 3D estimate, in that case sensor estimates can be combined (for instance in the case of GPS and INS).

Once a pose estimate is known, the data are registered on the Manifold. When this happens, the observers are notified. Examples of these observers are the behaviors, as described in Sec. 6. Other observers are the layers with are part of the user interface.

The other branch is the image processing performed on the camera images. The images can be processed on color. This is done to detect the victim [65], another robot[13] or the soccer landmarks[9]. As an example, an extended version of description of the victim detection from [65] is repeated here.

4.1 Victim detection

Until 2007 victims could be detected with a sort of RFID-tag. To make this sensor realistic, the RFID-tag did not provide ground truth, but had a certain chance on false positives and false negatives. To reduce the number of false negatives a victim detection based on skin detection is developed. A general 3D color histogram model will be constructed in which discrete probability distributions are learned [28]. Given skin and non-skin histograms based on training sets we can compute the probability that a given color value belongs to the skin and non-skin classes.

From this a skin pixel classifier is derived through the standard likelihood ratio approach [19]. A threshold based on the costs of false positives and false negatives forms the basis for the skin pixel classifier.

4.1.1 The color Model

We first construct a general color model from the generic training set using a histogram with 32 bins of size 8 per channel in the RGB color space. The histogram counts are converted into a discrete probability distribution $P(\cdot)$ in the usual manner: $P(rgb) = \frac{c[rgb]}{T_c}$ where $c[rgb]$ gives the count in the histogram bin associated with the RGB color triple rgb and T_c is the total count obtained by summing the counts in all of the bins.

4.1.2 Skin Detection

We derive a skin pixel classifier through the standard likelihood ratio approach [19]. Given skin and non-skin histograms we can compute the probability that a given color value belongs to the skin and non-skin classes: $P(rgb|skin) = \frac{s[rgb]}{T_s}, \quad P(rgb|\neg skin) = \frac{n[rgb]}{T_n}$ where $s[rgb]$ is the pixel count contained in bin rgb of the skin histogram, $n[rgb]$ is the equivalent count from the non-skin histogram, and T_s and T_n are the total counts contained in the skin and non-skin histograms, respectively.

[9] the `2010\assistance` branch

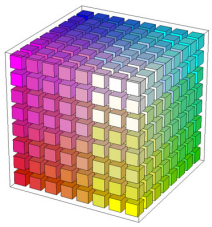


Figure 4: A 3-D color Histogram

Given a certain threshold, Θ, based on the costs of false positives and false negatives, a skin pixel classifier is constructed:

$$\frac{P(rgb|skin)}{P(rgb|\neg skin)} \geq \Theta \tag{1}$$

An example of this classifier, preliminary trained in the small world 'DM-VictimTest'[10] with only three victims, is given in figure 5. Because all three victims wore the same clothing, blue and white are still important components of this probability. Extending the training set with a wider variety of victims will reduce the influence of those colors, in favor of proper skin values.

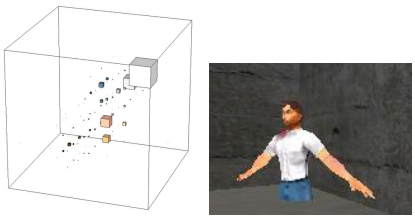


Figure 5: A plot of $\frac{P(rgb|skin)}{P(rgb|\neg skin)}$ derived from an environment of which the image to the right is a camera-image during positive VictSensor readings.

This classifier can be used to verify the artificial VictSensor readings, and to detect victims on larger distances and behind glass. This classifier can also be used to initiate a tracking algorithm based on color-histograms [72] to be able to cope with walking victims. The main aspect of this statistical method which make it so powerful is the fact that it is fast, compared with the notion that skin and non-skin histograms are quite distinct. In [28] it was shown that when the marginal distributions which result from integrating the 3-D histograms along green-magenta axis are compared, skin histograms show an obvious bias towards the red (figure 6).

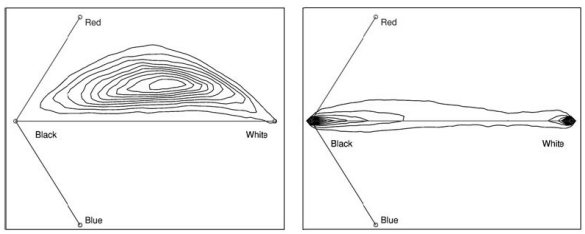


Figure 6: Two equiprobability contour plots for 2D-projections along the green-magenta axis of skin and non-skin models respectively.

The detection of victims based on shape is extensively described in Helen Flynn's thesis[15].

[10] An UT2004 based map available on `svn://u013154.science.uva.nl/Roboresc/Tijn/DM-VictTest_250.ut2`.

5 Simultaneous Localization and Mapping

Maps enable a robot to maintain an estimate of its physical surroundings and subsequently to keep track of its current location in the mapped environment. Thereby the map provides a spatial context for the interpretation of current and past observations and is the key enabler for higher level reasoning like exploration and coordination with team-members. Due to their central role in many aspects of a mobile robot's intelligent behavior, the capabilities exhibited by the chosen map representation are crucial to the successful operation of a team of autonomous robots.

In [40] we presented a sophisticated map representation that was specifically designed for use by teams of multiple robots. The map representation was inspired by the manifold from [26] and could be classified as a hybrid representation [53] which features both the strengths of a flexible topological graph and of a detailed occupancy grid. This section is a description which explains the approach for a more general artificial intelligence audience and is borrowed from unpublished work.

We used the presented approach for our participation in the Virtual Robots League of the RoboCup Rescue World Championships in 2006 [51]. There the data structure demonstrated scalability up to 8 robots. The scalability was achieved without sacrificing on other map aspects as accuracy and detail. We had maps that were accurate up to 2-20 centimeters and that were an order of magnitude more detailed than those of fellow competitors. This enabled us to win the Best Mapping Award that year [5].

In 2007 we participated again [69]. We deployed up to 6 robots which was the largest team deployed that year and our maps constantly received maximum or near maximum rewards on the aspects of 'Metric Quality', 'Skeleton Quality' and 'Utility' [11].

In this section the data structure of the manifold will be discussed in detail. Special attention will be paid to the features that facilitate multi-robot exploration.

5.1 Patches and Relations

The manifold is a hybrid map representation with a graph organization at the global level and small detailed metric maps at the local level. The vertices of this graph are also called the *patches* and the edges are referred to as the *relations* between these patches. Let Π denote the manifold, then we can define this as the set of all patches π and relations ϕ such that $\Pi = \{\{\pi\}, \{\phi\}\}$.

Each patch is of finite extent and defines a local planar coordinate system. A single patch stores a single laser range scan observation s together with the estimated global robot pose θ from where this scan was taken. A single scan as returned by the laser range sensor will consist of a set of n polar coordinates (α, d), which are easily translated into local (x, y) coordinates relative to the patch origin. So:

$$\pi = (\theta, s) : \theta = (x, y, \rho), s = \{(\alpha, d)^n\}$$

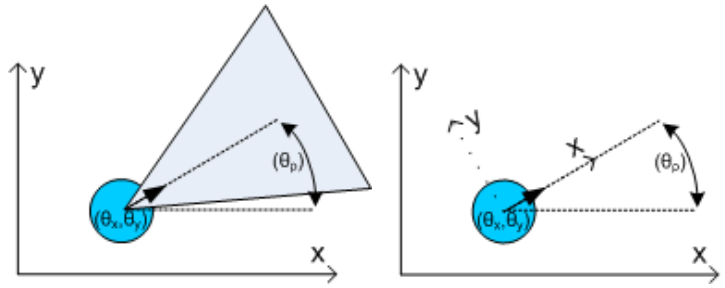
.

In effect, the patches discretize the full map into small, possibly overlapping, local, metric maps. The pose θ denotes the origin of the local coordinate frame and it provides the transformation from the global coordinate frame to the local measurement frame and vice versa. Let r_{π_a} be a robot pose estimate relative to patch π_a, then $\oplus$ is defined as the coordinate transformation operator that projects this pose estimate on the global frame and $\ominus$ as the inverse operator that projects it back to a patch-relative pose estimate [26]:

$$r_{global} = r_{\pi_a} \oplus \theta_a$$

[11] Unfortunatelly, the Mapping competition is no longer a separate part of the competition, because it is difficult to judge the quality in an objective way. See [3] and [70] for more details on evaluating map aspects.

$$r_{\pi_b} = r_{\pi_a} \oplus \theta_a \ominus \theta_b$$



(a) Patch as pose $\theta_{(x,y,\rho)}$ with scan (b) Patch as local coordinate system

Figure 7: Patches.

Relations connect the patches in the manifold and indicate navigability as they are typically constructed between consecutive robot poses. Every relation stores a Gaussian probability distribution over the estimated pose-difference $\Delta\theta_{ab}$ between two related patches π_a and π_b. This probability distribution with mean $\Delta\theta_{ab}$ and covariance matrix Σ_{ab} is estimated from the set of *pair-wise point-correspondences* between the two patches. Typically, the parameters of this probability distribution are estimated by a scan matcher [41].

The probability distribution can be stored by defining the relation ϕ_{ab} two patches π_a and π_b as:

$$\phi_{ab} = (\pi_a, \pi_b, \Delta\theta_{ab}, \Sigma_{ab})$$

. Figure 8 gives a schematic illustration of several consecutive robot poses and the manifold that could have been constructed along this trajectory.

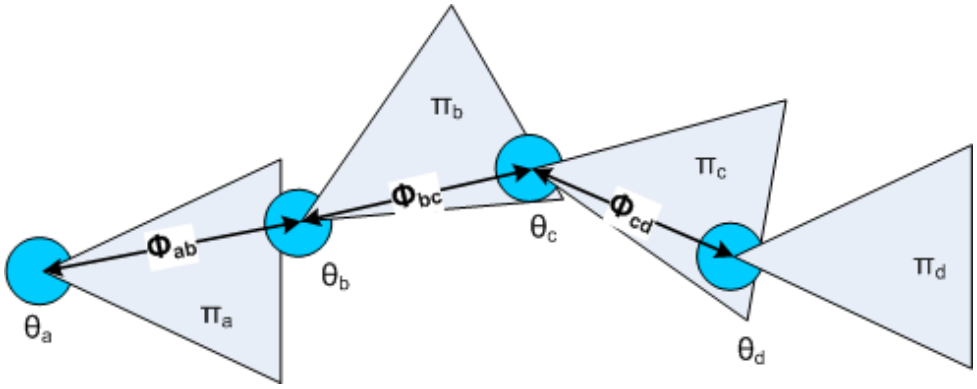


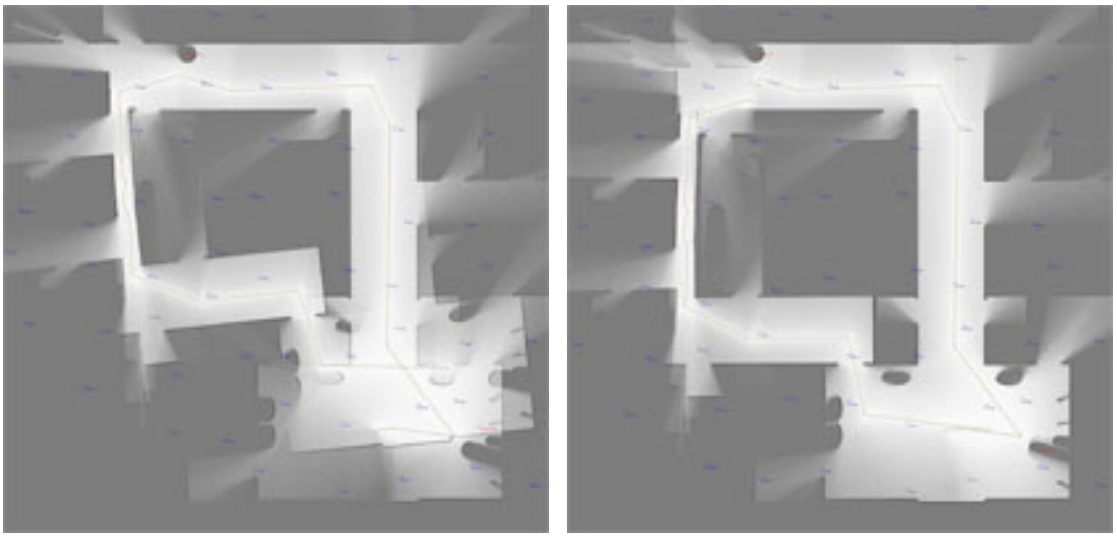
Figure 8: Schematic illustration of a manifold that could have been constructed along a short sample trajectory.

5.2 Mapping Operations

As the robot explores the environment the manifold will grow accordingly and map the visited areas. Patches are added to capture the local geometric properties of the environment and relations are inserted to store the navigable paths from one patch to the next.

All displacement information is estimated using an iterative closest point (ICP) scan matcher [41] that compares the current laser scan with laser scans recorded shortly before, stored in nearby patches of the graph. As long as the new range scan could be matched with sufficient

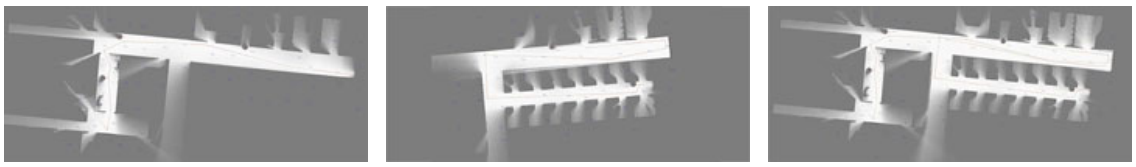
confidence the displacement information is only used to localize the robot. However, when this confidence drops below certain thresholds the new scan is considered relevant enough to memorize it. Hence a new patch is created that stores the scan and the uncertainty information is stored on a newly created relation. A new part of the map was learned.



(a) Before loop closure. (b) After loop closure

Figure 9: Loop-closing. The robot starts at the bottom right and moves up. Then the robot turns left several times until it returns to the bottom right, re-observes a particular landmark and closes the loop.

When new parts of the map start to overlap with previously mapped parts the scan matcher can also be used to determine correspondence between the overlapping regions. If the scan matcher is confident that these overlapping regions in fact map the same area, a loop closure algorithm can be triggered as in Figure 9. Similarly, when multiple robots hypothesize that they explored the same area their maps could be merged into one using the same procedure, see Figure 10 for an example.



(a) NorthEastern Map. (b) Eastern Map. (c) Merged Map.

Figure 10: Map merging. Two robots partly explore the same area (Northern and Eastern maps) which they merge after which they acquire the Merged map on the right.

5.3 Sharing Map Information between Multiple Robots

A key aspect of the manifold's design that enhances its scalability to teams of multiple robots is in how information is stored on the patches. All relevant information that a robot wishes to memorize is always stored in the local coordinate frames of the relevant patches. The consequence is that a patch thereby becomes a fully self-contained piece of information that can be shared easily and independently with team members.

Since also relations only store information about the *relative* displacement between patches, robots can also easily communicate partial maps involving multiple patches and the relations between them.

5.4 Integrating Maps from Other Robots

A single manifold can simultaneously serve multiple robots. In this case each robot starts on its own patch which it develops into a disconnected sub-graph as it explores and maps the environment. The fact that these sub-graphs are disconnected exactly represents the fact that initially the relative displacement of multiple robots may be unknown. As soon as two robots meet they can decide to align and merge their sub-graphs into one connected component as illustrated in Fig. 10.

A different configuration, but with the same underlying idea, is when robots not actually share the same instance of the manifold but communicate updates to each other. In this scenario each robot uses its own manifold in which it maintains a separate disconnected sub-graph for each team member.

It is interesting to note that robots can decide to keep track of all available information *without* actually merging the individual maps. So, a robot can have an approximate idea of where its team members currently are and where they are heading without having to risk polluting his own map by merging potentially incompatible maps. Merging can be delayed until enough certainty is acquired about the correspondence. Similarly, if a robot loses track of its location, e.g. because it fell downstairs or bumped into something unexpected, it can simply start a new disconnected sub-graph and merge this later when it re-establishes its location.

6 Behaviors

There are a wide variety of behaviors implemented. During the 2011 competition Julian de Hoog limited the choices during the competition to the four most relevant for the competition:

- **TeleOperation:** this behavior gives direct control of the robot via the buttons in the ActionController.
- **ConservativeTeleOp:** this behavior is equivalent with TeleOperation, with the exception that the robot will stop when it senses a wall in front of the robot. The stop can be overruled by pressing the forward button again.
- **FollowWaypoint:** this behavior works in concert with waypoints indicated on the map for this particular robot. The robot performs path-planning (breath-first) to the waypoint.
- **AutonomousExploration:** this behavior is extensively described in [69]. It distributes frontiers over robots based on the distance to this robot (costs) and the area beyond the frontier (gain). To calculate the distance it first makes an initial estimate on Euclidian distance, followed by a better estimate based path-planning (actually the same as used by the FollowWaypoint).

Next to those four behaviors, several more experimental behaviors are implemented. Examples are DeploymentBehavior, which was one of the challenges during the earlier competitions. The goal was to deploy a relay network as large as possible inside the devastated building. The robots have to stay in communication-range, although it is difficult to predict the distribution in advance, because the signal is not only a function of the distance, but also from the attenuation of the obstacles between the robots. Another behavior is ExploreTraversibility, which was the behavior developed during this study [56].

6.1 Motions

The behaviors are implemented by a number of motions, each representing a certain reaction to sensor events. When it is detected that the current reaction is not appropriate a transition to another motion is made. Motions could be reused in different behaviors, yet their behavior and transitions could be slightly different for each behavior. A way to organize this neatly is to combine them in a folder, as done for the Following behavior. In Fig. 11 the motions and transitions are indicated.

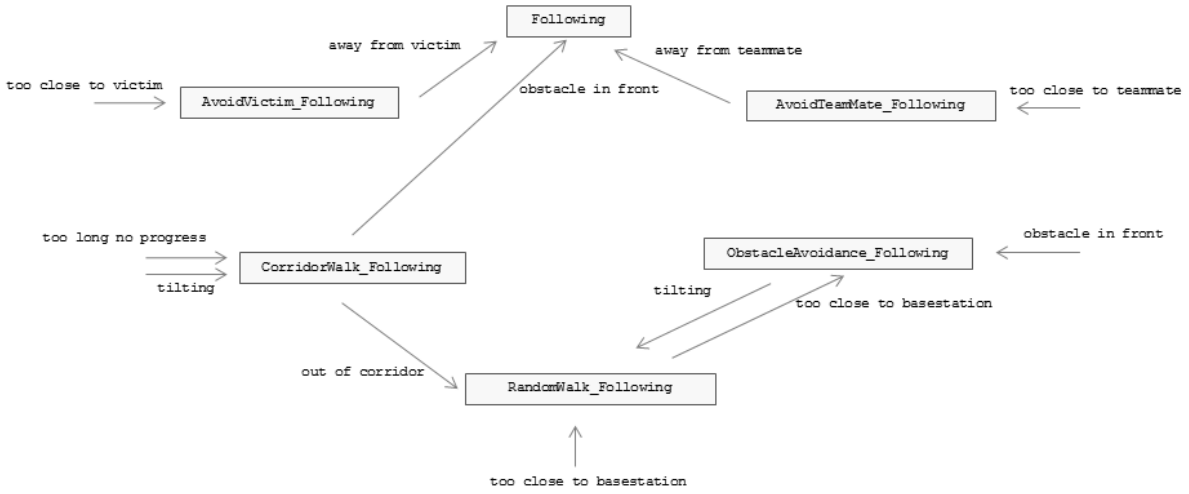


Figure 11: Indication of motions and transitions inside the FollowingBehavior.

The motions *RandomWalk* and *ObstacleAvoidance* have nearly the same rules, although the *RandomWalk* reacts on the Laser scan while *ObstacleAvoidance* reacts on the Sonar sensors. The motions *AvoidVictim* and *AvoidTeamMate* also have nearly the same rules and in addition have clear moment when they are activated and deactivated. The two motions *Following* and *CorridorWalk* are both high level motions. *Following* relies on path planning. When this model-based approach does not work, it falls back on a sensor-based approach; let the environment guide the robot for a while by following the walls.

7 Conclusion

The intention of this report is to give an overview of the code developed during the last six years. This report is not a reference manual, not every function is explained in detail. Yet, most function names are self-explaining and comments can be found throughout the code. The code is maintained in a repository, which allows finding back the developer and date, which allows to find more details in the corresponding logbook[12]. In addition, this document also is an entrance to all reports, papers, articles and theses[13], which describes the research that the leaded to the algorithms. It is the intention to maintain this document and to update it yearly with the latest developments inside the UvA Rescue Team.

[12] `http://www.jointrescueforces.eu/wiki/tiki-index.php?page=Research_Logs`
[13] Available for download on `http://www.jointrescueforces.eu/wiki/tiki-index.php?page=Publications`

References

[1] A. Abbo and S. Peelen, "Progressive Deepening for GameTrees: An application for RoboRescue", Bachelor's thesis, Universiteit van Amsterdam, June 2004.

[2] F. Alnajar, H. Nijhuis and A. Visser, "Coordinated action in a Heterogeneous Rescue Team", in "RoboCup 2009: Robot Soccer World Cup XIII", *Lecture Notes in Artificial Intelligence*, volume 5949, pp. 1–10, Springer, Heidelberg, February 2010, ISBN 978-3-642-11875-3.

[3] B. Balaguer, S. Balakirsky, S. Carpin and A. Visser, "Evaluating maps produced by urban search and rescue robots: lessons learned from RoboCup", *Autonomous Robots*, volume 27(4):pp. 449–464, November 2009.

[4] B. Balaguer, S. Carpin, S. Balakirsky and A. Visser, "Evaluation of RoboCup Maps", in "Proceedings of the 9th Performance Metrics for Intelligent Systems (PERMIS'09) workshop", September 2009.

[5] S. Balakirsky, S. Carpin, A. Kleiner, M. Lewis, A. Visser, J. Wang and V. A. Ziparo, "Towards heterogeneous robot teams for disaster mitigation: Results and Performance Metrics from RoboCup Rescue", *Journal of Field Robotics*, volume 24(11-12):pp. 943–967, November 2007, doi:10.1002/rob.20212.

[6] S. Balakirsky, S. Carpin and A. Visser, "Evaluation of The RoboCup 2009 Virtual Robot Rescue Competition", in "Proceedings of the 9th Performance Metrics for Intelligent Systems (PERMIS'09) workshop", September 2009.

[7] C. Bastiaan, "Virtual victims in USARSim", Bachelor's thesis, Universiteit van Amsterdam, June 2010.

[8] T. M. Cover and J. A. Thomas, *Elements of information theory*, Wiley-Interscience, New York, NY, USA, 1991.

[9] M. P. De Waard, "Combining RoboCup Rescue and XABSL", Bachelor's thesis, Universiteit van Amsterdam, June 2012.

[10] N. Dijkshoorn, "Simultaneous localization and mapping with the AR.Drone", Masters thesis, Universiteit van Amsterdam, July 2012.

[11] N. Dijkshoorn, H. Flynn, O. Formsma, S. van Noort, C. van Weelden, C. Bastiaan, N. Out, O. Zwennes, S. S. Otárola, J. de Hoog, S. Cameron and A. Visser, "Amsterdam Oxford Joint Rescue Forces - Team Description Paper - RoboCup 2011", in "Proc. CD of the 15th RoboCup International Symposium", June 2011.

[12] N. Dijkshoorn and A. Visser, "Integrating Sensor and Motion Models to Localize an Autonomous AR.Drone", *International Journal of Micro Air Vehicles*, volume 3(4):pp. 183–200, December 2011.

[13] A. Ethembabaoglu, "Active target tracking using a mobile robot in the USARSim", Bachelor's thesis, Universiteit van Amsterdam, June 2007.

[14] M. L. Fassaert, S. B. Post and A. Visser, "The common knowledge model of a team of rescue agents", in "1th International Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disaster", July 2003.

[15] H. Flynn, *Machine Learning Applied to Object Recognition in Robot Search and Rescue Systems*, Master's thesis, University of Oxford, September 2009.

[16] H. Flynn, J. de Hoog and S. Cameron, "Integrating Automated Object Detection into Mapping in USARSim", in "Proceedings of the International Conference on Intelligent Robots and Systems (IROS 2009), Workshop on Robots, Games, and Research: Success stories in USARSim", pp. 29–34, October 2009.

[17] O. Formsma, N. Dijkshoorn, S. van Noort and A. Visser, "Realistic Simulation of Laser Range Finder Behavior in a Smoky Environment", in "RoboCup 2010: Robot Soccer World Cup XIV", *Lecture Notes on Artificial Intelligence*, volume 6556, pp. 336–349, Springer, June 2011.

[18] U. Frese, "A Discussion of Simultaneous Localization and Mapping", *Autonomous Robots*, volume 20(1):pp. 25–42, 2006.

[19] K. Fukunaga, *Introduction to statistical pattern recognition (2nd ed.)*, Academic Press Professional, Inc., San Diego, CA, USA, 1990, ISBN 0-12-269851-7.

[20] J. de Hoog, *Role-Based Multi-Robot Exploration*, Ph.D. thesis, University of Oxford, May 2011.

[21] J. de Hoog, S. Cameron and A. Visser, "Robotic Search-and-Rescue: An integrated approach", in "Proc. of the Oxford University Computing Laboratory student conference 2008", Number RR-08-10 in OUCL, pp. 28–29, October 2008.

[22] J. de Hoog, S. Cameron and A. Visser, "Role-Based Autonomous Multi-Robot Exploration", in "Proceedings of the International Conference on Advanced Cognitive Technologies and Applications (Cognitive 2009)", November 2009.

[23] J. de Hoog, S. Cameron and A. Visser, "Autonomous Multi-Robot Exploration in Communication-Limited Environments", in "Proceedings of the 11th Conference Towards Autonomous Robotic Systems (Taros 2010)", Augustus/September 2010.

[24] J. de Hoog, S. Cameron and A. Visser, "Dynamic Team Hierarchies in Communication-Limited Multi-Robot Exploration", in "Proceedings of the IEEE International Workshop on Safety, Security and Rescue Robotics (SSRR 2010)", July 2010.

[25] J. de Hoog, S. Cameron and A. Visser, "Selection of Rendezvous Points for Multi-Robot Exploration in Dynamic Environments", in "International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)", May 2010.

[26] A. Howard, G. S. Sukhatme and M. J. Matarić, "Multi-Robot Mapping using Manifold Representations", *Proceedings of the IEEE - Special Issue on Multi-robot Systems*, 2006.

[27] M. Jankowska, *A Hough Transform Based Approach to Map Stitching*, Master's thesis, University of Oxford, September 2009.

[28] M. J. Jones and J. M. Rehg, "Statistical Color Models with Application to Skin Detection.", *International Journal of Computer Vision*, volume 46(1):pp. 81–96, 2002.

[29] S. Katt, "Introducing movements and animations to virtual victims in USARSim", Bachelor's thesis, Universiteit van Amsterdam, June 2012.

[30] G. E. Maillette de Buy Wenniger, T. Schmits and A. Visser, "Identifying Free Space in a Robot Bird-Eye View", in "Proceedings of the 4th European Conference on Mobile Robots (ECMR 2009)", September 2009.

[31] Q. Nguyen, "A Color Based Range Finder for an Omnidirectional Camera", Bachelor's thesis, Universiteit van Amsterdam, June 2009.

[32] Q. Nguyen and A. Visser, "A Color Based Rangefinder for an Omnidirectional Camera", in "Proc. IROS Workshop on Robots, Games, and Research: Success stories in USARSim", (edited by S. Balakirsky, S. Carpin and M. Lewis), pp. 41–48, 2009, ISBN 978-1-4244-3804-4.

[33] S. van Noort, "Validation of the dynamics of an humanoid robot in USARSim", Master's thesis, Universiteit van Amsterdam, May 2012.

[34] S. van Noort and A. Visser, "Extending Virtual Robots towards RoboCup Soccer Simulation and @Home", in "Proceedings of the 16th RoboCup Symposium", June 2012, to be published in the Springer Lecture Notes on Artificial Intelligence series.

[35] S. van Noort and A. Visser, "Validation of the dynamics of an humanoid robot in USARSim", in "Proceedings of Performance Metrics for Intelligent Systems Workshop (PerMIS12)", March 2012.

[36] F. A. Oliehoek and A. Visser, "A hierarchical model for decentralized fighting of large scale urban fires", in "International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)", May 2006.

[37] F. A. Oliehoek and A. Visser, *Interactive Collaborative Informations Systems*, *Studies in Computational Intelligence*, volume 281, chapter A Decision-Theoretic Approach to Collaboration: Principal Description Methods and Efficient Heuristic Approximations, pp. 87–124, Springer-Verlag, Berlin Heidelberg, March 2010, ISBN 978-3-642-11687-2, doi: 10.1007/978-3-642-11688-9_4.

[38] N. Out, "Virtual radar sensor for USARSim", Bachelor's thesis, Universiteit van Amsterdam, June 2010.

[39] M. Pfingsthorn, B. Slamet, A. Visser and N. Vlassis, "UvA Rescue Team 2006; RoboCup Rescue - Simulation League", in "Proc. CD of the 10th RoboCup International Symposium", 2006.

[40] M. Pfingsthorn, B. A. Slamet and A. Visser, "A Scalable Hybrid Multi-Robot SLAM Method for Highly Detailed Maps", in "RoboCup 2007: Robot Soccer World Cup XI", *Lecture Notes on Artificial Intelligence*, volume 5001, pp. 457–464, Springer-Verlag, July 2008.

[41] S. T. Pfister, *Algorithms for Mobile Robot Localization and Mapping, Incorporating Detailed Noise Modelling and Multi-scale Feature Extraction*, Ph.D. thesis, California Institute of Technology, April 2006.

[42] S. B. Post, M. L. Fassaert and A. Visser, "Reducing the communication for multiagent coordination in the RoboCupRescue Simulator", in "7th RoboCup International Symposium, Padua, Italy", July 2003.

[43] S. B. Post, M. L. Fassaert and A. Visser, "The high-level communication model for multiagent coordination in the RoboCupRescue Simulator", in "RoboCup 2003: Robot Soccer World Cup VII", *Lecture Notes on Artificial Intelligence*, volume 3020, pp. 503–509, Springer, June 2004.

[44] S. B. M. Post and M. L. Fassaert, *A communication and coordination model for 'RoboCupRescue' agents*, Master's thesis, Universiteit van Amsterdam, June 2004.

[45] S. B. M. Post, M. L. Fassaert and A. Visser, "The high-level communication model for multiagent coordination in the RoboCupRescue Simulator", in "7th RoboCup International Symposium", (edited by D. Polani, B. Browning, A. Bonarini and K. Yoshida), *Lecture Notes on Artificial Intelligence*, volume 3020, pp. 503–509, Springer-Verlag, 2004.

[46] S. Roebert, "Creating a bird-eye view map using an omnidirectional camera", Bachelor's thesis, Universiteit van Amsterdam, June 2008.

[47] S. Roebert, T. Schmits and A. Visser, "Creating a Bird-Eye View Map using an Omnidirectional Camera", in "Proceedings of the 20th Belgian-Netherlands Conference on Artificial Intelligence (BNAIC 2008)", October 2008.

[48] R. Rozeboom, "Navigating using a radar sensor in USARSim", Bachelor's thesis, Universiteit van Amsterdam, June 2012.

[49] T. Schmits, *Development of a Catadioptric Omnidirectional Camera for the USARSim Environment*, Master's thesis, Universiteit van Amsterdam, June 2008.

[50] T. Schmits and A. Visser, "An Omnidirectional Camera Simulation for the USARSim World", in "RoboCup 2008: Robot Soccer World Cup XII", (edited by L. Iocchi, H. Matsubara, A. Weitzenfeld and C. Zhou), *Lecture Notes in Artificial Intelligence*, volume 5339, pp. 296–307, Springer, June 2009.

[51] B. A. Slamet and M. Pfingsthorn, *ManifoldSLAM: a Multi-Agent Simultaneous Localization and Mapping System for the RoboCup Rescue Virtual Robots Competition*, Master's thesis, Universiteit van Amsterdam, December 2006.

[52] R. Sobolewski, *Machine Learning for Automated Robot Navigation in Rough Terrain*, Master's thesis, University of Oxford, September 2009.

[53] S. Thrun, "Robotic Mapping: A Survey", in "Exploring Artificial Intelligence in the New Millenium", (edited by G. Lakemeyer and B. Nebel), Morgan Kaufmann, 2002.

[54] M. van Ittersum, Xingrui-Ji, L. Gonzalez and L. Stancu, "Natural Boundaries", Project Report, Universiteit van Amsterdam, February 2007.

[55] M. van der Veen, "Optimizing Artificial Force Fields for Autonomous Drones in the Pylon Challenge using Reinforcement Learning", Bachelor's thesis, Universiteit van Amsterdam, July 2011.

[56] M. van der Velden, W. Josemans, B. Huijten and A. Visser, "Application of Traversability Maps in the Virtual Rescue competition", in "RoboCup IranOpen 2010 Symposium (RIOS10)", April 2010.

[57] A. Visser, "A survey of the architecture of the communication library LCM for the monitoring and control of autonomous mobile robots", Technical Report IAS-UVA-12-01, Informatics Institute, University of Amsterdam, The Netherlands, October 2012.

[58] A. Visser, G. E. M. de Buy Wenniger, H. Nijhuis, F. Alnajar, B. Huijten, M. van der Velden, W. Josemans, B. Terwijn, C. Walraven, Q. Nguyen, R. Sobolewski, H. Flynn, M. Jankowska and J. de Hoog, "Amsterdam Oxford Joint Rescue Forces - Team Description Paper - RoboCup 2009", in "Proc. CD of the 13th RoboCup International Symposium", July 2009.

[59] A. Visser, N. Dijkshoorn, S. van Noort, O. Zwennes, M. de Waard, S. Katt and R. Rozeboom, "UvA Rescue - Team Description Paper - RoboCup 2012", June 2012.

[60] A. Visser, N. Dijkshoorn, M. van der Veen and R. Jurriaans, "Closing the gap between simulation and reality in the sensor and motion models of an autonomous AR.Drone", in "Proceedings of the International Micro Air Vehicle Conference and Flight Competition (IMAV11)", September 2011.

[61] A. Visser and J. de Hoog, "Amsterdam Oxford Joint Rescue Forces - Realistic Simulations to aid research and education in advanced Robot Control algorithms", in "Proc. of the Scientific ICT Research Event Netherlands (SIREN 2008)", p. 22, September 2008.

[62] A. Visser, Q. Nguyen, B. Terwijn, M. Hueting, R. Jurriaans, M. van de Veen, O. Formsma, N. Dijkshoorn, S. van Noort, R. Sobolewski, H. Flynn, M. Jankowska, S. Rath and J. de Hoog, "Amsterdam Oxford Joint Rescue Forces - Team Description Paper - RoboCup 2010 and Iran Open", in "Proc. CD of the 14th RoboCup International Symposium", July 2010.

[63] A. Visser, G. Pavlin, S. P. van Gosliga and M. Maris, "Self-organization of multi-agent systems", in "Proc. of the International workshop Military Applications of Agent Technology in ICT and Robotics", November 2004.

[64] A. Visser, T. Schmits, S. Roebert and J. de Hoog, "Amsterdam Oxford Joint Rescue Forces - Team Description Paper - RoboCup 2008", in "Proc. CD of the 12th RoboCup International Symposium", July 2008.

[65] A. Visser, B. Slamet, T. Schmits, L. A. González Jaime and A. Ethembabaoglu, "Design decisions of the UvA Rescue 2007 Team on the Challenges of the Virtual Robot competition", in "Proc. 4th International Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disaster", pp. 20–26, July 2007.

[66] A. Visser and B. A. Slamet, "Balancing the Information Gain Against the Movement Cost for Multi-robot Frontier Exploration", in "European Robotics Symposium 2008", Springer Tracts in Advanced Robotics, pp. 43–52, Springer-Verlag, February 2008.

[67] A. Visser and B. A. Slamet, "Including communication success in the estimation of information gain for multi-robot exploration", in "6th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks and Workshops (WiOPT 2008)", pp. 680–687, IEEE Publishing, April 2008, doi:10.1109/WIOPT.2008.4586160.

[68] A. Visser, B. A. Slamet and M. Pfingsthorn, "Robust Weighted Scan Matching with Quadtrees", in "Proc. of the 5th International Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disaster (SRMED 2009)", July 2009.

[69] A. Visser, Xingrui-Ji, M. van Ittersum, L. A. González Jaime and L. A. Stancu, "Beyond frontier exploration", in "RoboCup 2007: Robot Soccer World Cup XI", *Lecture Notes in Artificial Intelligence*, volume 5001, pp. 113–123, Springer-Verlag, July 2008.

[70] C. Walraven, "Using path planning to grade the quality of a mapper", Bachelor's thesis, Universiteit van Amsterdam, June 2009.

[71] B. L. Wellman, J. de Hoog, S. Dawson and M. Anderson, "Using Rendezvous to Overcome Communication Limitations in Multirobot Exploration", in "Proceedings of SMC (IEEE International Conference on Systems, Man and Cybernetics)", October 2011.

[72] Z. Zivkovic and B. Kröse, "An EM-like algorithm for color-histogram-based object tracking", in "IEEE Conference on Computer Vision and Pattern Recognition", volume 1, pp. 798–803, June 2004.

[73] O. Zwennes, "Adaptive Indoor Map Generator for USARSim", Bachelor's thesis, Universiteit van Amsterdam, June 2011.

[74] O. Zwennes, A. Weiss and A. Visser, "Adapting the mapping difficulty for the automatic generation of rescue challenges", in "RoboCup IranOpen 2012 Symposium (RIOS12)", April 2012.

Acknowledgements

The structure of this document is inspired by the document published by the B-Human team and its predecessor the German Team. The code described in this report is build as a joint effort by many students and reseachers from the University of Amsterdam and Oxford, who all should receive a part of the credit.

IAS reports

This report is in the series of IAS technical reports. The series editor is Bas Terwijn (B.Terwijn@uva.nl). Within this series the following titles appeared:

A. Visser *A survey of the architecture of the communication library LCM for the monitoring and control of autonomous mobile robots* Technical Report IAS-UVA-12-01, Informatics Institute, University of Amsterdam, The Netherlands, October 2012.

C. Dimitrakakis *Bayesian variable order Markov models: Towards Bayesian predictive state representations* Technical Report IAS-UVA-09-04, Informatics Institute, University of Amsterdam, The Netherlands, September 2009.

C. Dimitrakakis and C. Mitrokotsa *Statistical decision making for authentication and intrusion detection* Technical Report IAS-UVA-09-03, Informatics Institute, University of Amsterdam, The Netherlands, August 2009.

P. de Oude and G. Pavlin *Dependence discovery in modular Bayesian networks* Technical Report IAS-UVA-09-02, Informatics Institute, University of Amsterdam, The Netherlands, June 2009.

C. Dimitrakakis *Complexity of stochastic branch and bound for belief tree search in Bayesian reinforcement learning* Technical Report IAS-UVA-09-01, Informatics Institute, University of Amsterdam, The Netherlands, April 2009.

All IAS technical reports are available for download at the ISLA website, http://isla.science.uva.nl/node/52.