View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Ghent University Academic Bibliography

# Subsumption Architecture for Enabling Strategic Coordination of Robot Swarms in a Gaming Scenario

Anna Hristoskova, Enric Junqué de Fortuny, and Filip De Turck

Department of Information Technology, Ghent University - IBBT,
Gaston Crommenlaan 8 bus 201, 9050 Ghent, Belgium
{anna.hristoskova,filip.deturck}@intec.ugent.be
{enric.junquedefortuny}@ugent.be

**Abstract.** The field of swarm robotics breaks away from traditional research by maximizing the performance of a group - swarm - of limited robots instead of optimizing the intelligence of a single robot. Similar to current-generation strategy video games, the player controls groups of units - squads - instead of the individual participants. These individuals are rather unintelligent robots, capable of little more than navigating and using their weapons. However, clever control of the squads of autonomous robots by the game players can make for intense, strategic matches.
The gaming framework presented in this article provides players with strategic coordination of robot squads. The developed swarm intelligence techniques break up complex squad commands into several commands for each robot using robot formations and path finding while avoiding obstacles. These algorithms are validated through a 'Capture the Flag' gaming scenario where a complex squad command is split up into several robot commands in a matter of milliseconds.

**Keywords:** Swarm robotics, Subsumption, Robot behaviours, Strategic control, Robot formations, Path finding

## 1 Introduction

The increasing amount of robotics researchers and technology developers results in new innovations and opportunities attracting attention outside the factory. Korea is taking the lead in promoting the use of robots for service applications, such as elderly care. The United States employs robots to assist soldiers on the battlefield. Robots are manufacturing solar panels for European companies. Projects where robots collaborate to successfully complete tasks are becoming increasingly common. An indicator of this rise in interest is the growing number of challenges, leagues and participants of the RoboCup initiative [1, 2]. It promotes robotics research by providing appealing scenarios such as robot soccer.

The idea of 'robot gaming' is not a new one. In the past, however, the application of this concept has been rather limited. Current robot games involve individual robots fighting each other in a last-robot-standing competition. Others are actually robot building and programming competitions, rather than true games. In swarm robotics, the challenge is to solve problems by using numerous simple robots by maximizing the performance of the collective behaviour of the swarm. These robots run on limited hardware supporting basic behaviour limiting their AI. Few swarm robotic systems use small finite state machine as the core controlling mechanism of the individual robots. This is due to the fact that most of them are inspired by the behaviour of insects having limited intelligence.

This article presents a framework enabling game players with strategic control over robot squads. The player has access to a gaming interface managing the health and power of his robots and the coordination of the squads. A swarm intelligence component is distributed between the central framework and the individual robots supporting dynamic real-time autonomous robot movement, robot formations, path finding, sensor reading, event detection and obstacle avoidance. Novelty is the implemenation of a subsumption architecture supporting the swarm intelligence. It decomposes a complex squad behaviour into many basic robot behaviours which are organized into layers and activated based on priority. Validation of the proposed framework for a 'Capture the Flag' scenario shows that the swarm intelligence is able to split a single strategic squad command into several robot commands in 570 ms.

The remainder of the paper is structured as follows. Section 2 presents the current research in the field of swarm robotics. Section 3 elaborates on the developed framework with special attention on the swarm intelligence in Section 4. The swarm behaviour is evaluated for a gaming scenario in Section 5. Finally, the main conclusions and future improvements are drawn in Section 6.

## 2 Related Work

Platforms requiring the simultaneous achievement of complex tasks focus on techniques related to multi-robot coordination and task allocation. RoboSwarm [3] supports global behaviour through centralized robot orchestration consisting of task planning and allocation using bidding mechanisms between robots. The distributed approach of I-Swarm [4] is based on collective behaviours observable in honeybees. Instead of high-level robot-to-robot communication a network of weak robot-to-robot interactions (collisions) leads to specific spatial constellations that promote a collective decision. Another insect-inspired (ants) project is Swarm-bots where teams of robots overcome challenges such as object movement, path formation and hole avoidance by autonomously attaching to each other and moving around in coordination [5–7]. Though these s-bots do not talk among themselves, they receive low-level signals - such as individual push and pull forces - allowing coordinated group movement. The process of self-assembling is governed by the attraction and the repulsion among s-bots, and between s-bots and other objects. The successor to the Swarm-bots project, Swarmanoid, supports self-assembly through the training of artificial neural networks in order to transport an object [8]. The project focuses on the creation of three kinds of robots;

eye-bots, hand-bots, and foot-bots. While the foot-bots move back and forth between a source and a target location, the eye-bots are deployed in stationary positions against the ceiling, with the goal of guiding the foot-bots [9].

In [10] the construction of a world model plays essential role for determining the positions of robot players on a soccer field, merging observations from vision, messages from teammates, and odometry information from motion. The position estimates are used by different behaviours to decide the robot's actions. These behaviours include skills (low-level robot abilities such as kicking the ball), tactics (behaviour of a single robot), and plays (coordination of the team) [11].

The framework in this article is based on the announce/listen metaphor, found in current routing protocols. A robot subscribes to and publishes messages on a specific topic/channel. Novelty is the implementation of a subsumption architecture decomposing a complicated swarm behaviour into many simple behaviours for each individual robot. It is based on the sliding autonomy [12] technique which enables humans to interact and take over control of a given subtask, while the rest of the system operates autonomously. Interventions include robots requesting help when a failure occurs or a user requesting robot assistance with some task. Studies show that sliding autonomy provides an increase in efficiency of 30-50% over teleoperation together with an increase in reliability of up to 13% over autonomous operation depending on the user's ability [13].

## 3 Framework Design and Implementation

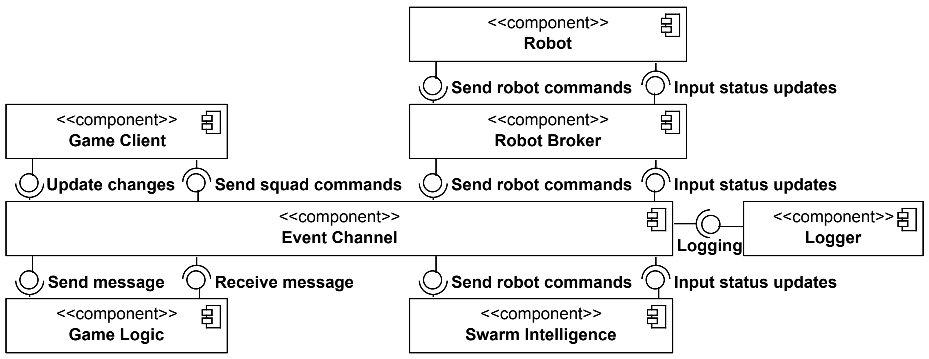

**Fig. 1.** Architecture of the publish-subscribe gaming framework showing the communication of the components through the Event Channel.

The proposed framework should be able to adapt dynamically to changing number of robots, players and most importantly messages as communication is a key concept required for the intelligent coordination of robot swarms. Therefore a Publisher-Subscriber pattern extended with an Event Channel is developed, as presented in Figure 1. Eventing allows for a scalable architecture processing and forwarding events generated by the Game Client, robots, Game Logic and Swarm Intelligence (all discussed in the following paragraphs). This results in a dynamic real-time control of the robot squads.

**The Event Channel** is responsible for the communication between the different components. As performance is a key issue ActiveMQ [14] is adopted which is designed for high performance clustering, client-server, and peer-based communication. It is an open sourced implementation of JMS 1.1 as part of J2EE 1.4 and supports a variety of cross language clients and protocols such as Java, C, C++, C#, Ruby, Perl, Python, PHP.

**The Game Client** connects the player to the game capturing and transmitting player actions (e.g. robot, squad and game configuration, chat sessions, strategic squad commands, status updates, monitoring messages, music) and visualizing the battlefield. This is supported by a PAC-pattern as a hierarchical structure of agents, each consisting of a triad of Presentation, Abstraction and Control. The agents are responsible for the players' actions communicating with each other through the Control. The Abstraction retrieves and processes the actions and the Presentation formats the visual and audio presentation of the battlefield.

**The Game Logic** sets up a game by connecting players with each other and with their robots. It manages the battlefield map, player and robot data, game settings and status, and player-squad communication through the realization of the player commands by the Swarm Intelligence. Several Game Logic modules can be connected to each other handling an increasing load of players and robots.

**The Swarm Intelligence** contains the AI of the framework, safe from the Robot Intelligence on the robots. It is presented in detail in Section 4.

**The Robot** has a layered design in Figure 2 keeping hardware dependencies local. The upper layer contains the Robot Intelligence, sensors and actuators. The Robot Intelligence acts as the "brain" of the robot interpreting orders from the Swarm Intelligence, gathering information from its sensors and deciding which sequence of actions has to be performed to execute the orders (detailed description provided in Section 4.4). The sensors and actuators control the robot's hardware. The Data Model prevents illegal robot operations by following the rules defined by the players at the start of the game. It controls the robot's vital parameters such as health, firepower, and provides the Robot Intelligence with a high-level abstraction of the robot's sensors, actuators and its communication interface. This Communication layer sends and receives messages and commands from and to other robots and the Game Logic through the Robot Broker.

**The Logger** monitors the game actions through the use of the open source program log4j [15]. It enables detection of conflicts and determines their source.

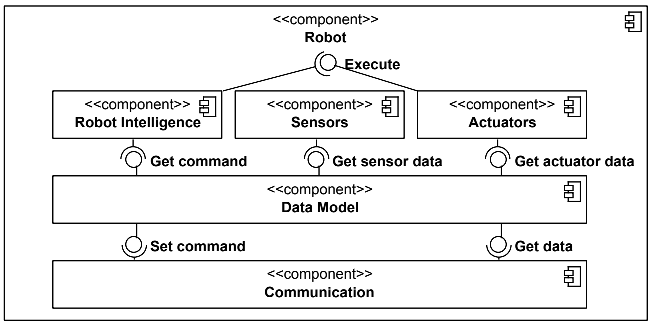

**Fig. 2.** The Layered Pattern of the Robot.

## 4 Detailed View on the Robot and Swarm Intelligence

While designing the Swarm Intelligence two main resource-related obstacles arise when working with limited devices such as robots. On one hand it is undesirable to have a huge amount of state information on each robot due to limited memory and communication overhead. Furthermore, not all robots posses enough computational resources required to perform advanced swarm behaviours. Therefore the designed architecture employs the **sliding autonomy** principle that is split into two distinct parts working together: reflexive behaviours or simple commands reside on the robot itself (**Robot Intelligence**), whereas complex swarm coordination computations are 'outsourced' to the server (**Server-side Swarm Intelligence**). The outcome is an architecture where the amount of intelligence residing on the robots is adaptable to their hardware capabilities. Before discussing them in detail, we will first focus on a description of the subsumption principle and path finding in combination with robot formations.

### 4.1 Subsumption and Behavioural Layers

The core of the Robot and Swarm Intelligence is built using the **subsumption architecture** in Figure 3. Subsumption is a way of decomposing a complicated intelligent behaviour into many simple behavioural modules which in turn are organized into layers. An Arbitrator activates one and only one behaviour having the highest priority when a choice between several behaviours is presented.

The cooperating parts on the server (**Swarm Intelligence**) and on the robot (**Robot Intelligence**) consist of different behavioural layers, stacked on top of each other. A user can easily add a behaviour by implementing the provided interface and adding it to the Arbitrator who will then include the behaviour in the arbitration procedure. The following main behavioural layers are defined:

- *Squad behaviours* have the lowest priority and are usually sequences of commands, performed by a squad of robots as a group. They are always initiated by a player command and processed at the server.

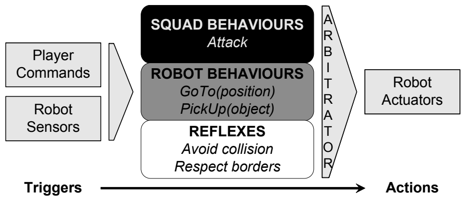

**Fig. 3.** Behavioural layers.

– *Robot behaviours* consist of basic individual robot commands such as GoTo(x,y), PickUp(object) and are a resultant of squad behaviours.
– *Reflexes* have the highest priority. Examples include last-minute collision avoiding and staying within map boundaries.

## 4.2 Formations

A critical part in any robotics application is path finding. In the presented architecture, the $A^*$-algorithm[1] is adopted for swarm navigation on the battlefield. Swarm navigation coordinates a group of robots to get into and maintain a formation with a certain shape [16, 17]. The Swarm Intelligence guides the robots into composing several formations depending on the situation at hand. The following are provided (Robot$_i$ with diameter $d$, number of robots $n$):

– Line (horizontal):

$$\begin{aligned}\text{Destination D} &= (D_x, D_y)\\ \text{Robot}_{i,x} &= D_x + d(i - \frac{n-1}{2})\\ \text{Robot}_{i,y} &= D_y\end{aligned}$$

– Circle for defending objects:

$$\begin{aligned}\text{Destination D} &= (D_x, D_y)\\ \text{Radius R} &= nd\\ \text{Robot}_{i,x} &= D_x + R\cos\left(\frac{i\,360^\circ}{n}\right)\\ \text{Robot}_{i,y} &= D_y + R\sin\left(\frac{i\,360^\circ}{n}\right)\end{aligned}$$

– Half Circle for attacking enemy robots/squads:

$$\begin{aligned}\text{Destination D} &= (D_x, D_y)\\ \text{Radius R} &= nd\\ \text{Robot}_{i,x} &= D_x + R\cos\left(\frac{i\,270^\circ}{n}\right)\\ \text{Robot}_{i,y} &= D_y + R\sin\left(\frac{i\,270^\circ}{n}\right)\end{aligned}$$

– (Inverse) Wedge: combination of two diagonal line formations.

[1]source: http://www.gamegardens.com/

### 4.3 Server-side Swarm Intelligence

During game flow a player orders a robot squad to execute a certain high-level command (e.g. "Attack!", "Defend the base!") through the Game Client. These commands are posted on the Event Channel and processed by the Swarm Intelligence on the server before being passed on to the respective squads. A *Squad Arbitrator*, which acts as a squad leader decision intelligence, activates the correct *Squad Behaviour*. This behaviour breaks down the high-level commands into *Robot Behaviours*, smaller *(Sequence)Commands*, for each individual squad member. The following main *Squad Behaviours* are defined:

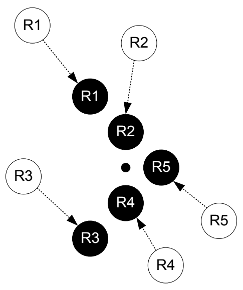

**Fig. 4.** Gathering robots in a wedge formation at a central point (black dot). Before gathering (white) and after gathering (black).

– *Gather:* The gather mechanism is presented in Figure 4. When a squad is initially created or is dispersed, the robots need to gather in their respective positions in a chosen formation. This is achieved by sending all robots to a central formation point. When no central point has been given, a centroid is calculated. Hereafter each robot is assigned a point in the final formation (based on minimal distance) and a path is calculated using the $A^*$-algorithm:
  1. $D(x, y)$ = central point.
  2. Calculate the best path $\mathcal{P}$ to $D(x, y)$ using $A^*$.
  3. Convert $\mathcal{P}$ into $n$ turn-points $P_i(x_i, y_i)$.
  4. Create sequence of commands using the points from Step 3.

$$\text{SequenceCommand} = \{\text{GoTo}(x_1, y_1), \text{GoTo}(x_2, y_2), ..., \text{GoTo}(x_n, y_n)\}$$

  5. Send *SequenceCommand* to the robots.
– *GoTo(x,y):* moves a group of robots from their respective locations to the given coordinates. First the robots in the squad are gathered in formation. Following the path $\mathcal{P}$ is calculated for the squad as a whole since moving in a straight line could lead to collisions. The player can also define the given path using click and drag or just individual commands.
– *Defend(object):* consists of:
  1. Get object's coordinates as $P(x, y)$.

  2. GoTo($x, y$) using circle or half-circle (moving object) formation.
- *Attack(object):* similarly to *Defend(object)* it consists of:
  1. Get object's coordinates as $P(x, y)$.
  2. GoTo($x, y$) using half-circle formation.

  A problem that might occur is that targets will often be moving. This is solved by recalculating the path which is fairly efficient given that only the action-radius of the target object has to be taken into account. On receiving a movement of the target:
  1. Do a partial $A^*$ on the battlefield map given the old data and the current position of the attacker.
  2. Update the old data entry on the map to the recent version.
  3. Send the new command to the robots.
- *Shoot(object):* is automatically done when an enemy robot is in range. The attacker 'shoots' at a preconfigured shooting rate with preconfigured damage. This damage is deducted from the enemy robot's health.
- *Avoiding Collisions:* For static objects on the battlefield map $A^*$ is used to avoid collisions. Furthermore, close range collision prevention is covered in the robot (*Reflex Behaviour*). It is however important to realize the dangers of moving objects such as enemy robots. These are not accounted for in path-finding algorithms due to exponentially high complexity. A mitigation strategy of the *Squad Behaviour* is to check for inbound collisions in a specific range for each squad. Once a possible inbound collision has been found, the squad's path is adjusted (Figure 5).

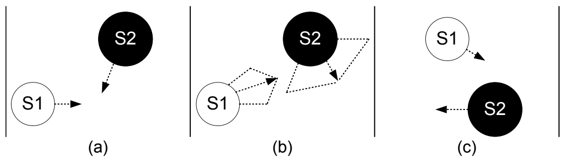

**Fig. 5.** Avoid collisions for dynamic objects Squad1 and Squad2; (a) a collision situation occurs, (b) mitigation vectors are calculated and commands are adjusted, (c) avoidance commands are performed, previous path is resumed.

### 4.4 Robot Intelligence

The robot is where all behaviours resulting from the *Squad Behaviours* of the server-side Swarm Intelligence eventually end up. It also adopts the subsumption architecture consisting of *SequenceCommands*, *Commands* and *Reflexes*. *Reflexes* are instantly activated and always get control when required. The following in Table 1 are defined:

- *Avoiding collisions*: Although the server-side Swarm Intelligence calculates a big part of path-finding and collision avoidance some collisions could still

occur. Therefore a basic collision behaviour for the individual robot is necessary in order to prevent any unwanted collisions or even possible damage.
- *Respecting borders*: Robots should not move beyond the gaming area or ignore physical map boundaries, such as water, while executing commands.

**Table 1.** Avoiding collisions and Respecting borders behaviours

| **Name** | Avoid collision | | Respect border |
|---|---|---|---|
| **Trigger** | Collision detected by the touch sensors (front and rear) | Object with $d(R,O) < d_{crit}$ detected by the sonar (viewing angle $\alpha$) | Border-line detected by the infrared sensor (front) |
| **Behaviour** | Turn the robot $90°$ away from the point of contact. | Turn the robot $90°$ and move forward for a length of $d_{crit}\tan(\alpha)$. | Turn the robot $90°$ away from the border. |

*Commands* (Table 2) received from the server-side Swarm Intelligence are performed on exactly one robot. They can be interrupted by *Reflexes* or overridden by new commands. There is always at most one command present in the robot. If a complex behaviour using multiple commands is required (e.g. Attack(object)), a *SequenceCommand* is used.

**Table 2.** Defined basic *Command behaviour*s.

| Name | Command behaviour |
|---|---|
| Beep | Produces a short beep, useful for shooting at enemies and testing. |
| Rotate($\alpha$) | Rotates the robot over $\alpha$-degrees. |
| GoTo(x,y) | Rotates the robot and moves it in a straight line to the given point. |
| Shoot(enemy) | Shoots at an enemy robot at a preconfigured shooting rate and damage. |

## 5 Validation of the Swarm Intelligence

The implemented swarm gaming[2] framework is executed using Lego Mindstorms NXT [18], a robotics toolkit for building different robots. Building instructions for 4 main models (Shooterbot, Colour Sorter, Alpha Rex, Robogator) ranging in complexity are supplied. All measurements are performed on a DELL Latitude E5400 notebook with 2.40 GHz Intel Core 2 Duo, and 4 GB RAM.

During normal game flow the game and robots (health power based on budget, squad division, robot type: scout, tank, infantry, amphibian) are configured. Several scenarios, such as 'Capture the Flag' and 'Death Match', are executed depending on the players' preferences. Measurements of the Swarm Intelligence

[2]http://ciri.be/blog/?p=234

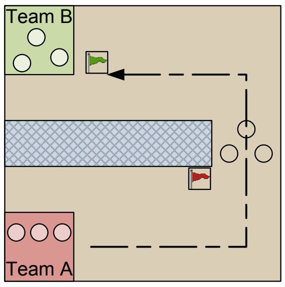

**Fig. 6.** 'Capture the Flag' battlefield including a water obstacle between the two teams.

are performed on squad A of 3 robots capturing the flag of an enemy team B. Figure 6 displays the robot positions and path taken in wedge formation by the bottom Team A. Player A selects squad Team A and indicates flag B. Recognized as an enemy flag by the Swarm Intelligence, this results in an GoTo(flag B) squad command. The *Squad Behaviour* breaks the command up into several (in this case 10 GoTo and a PickUp(flag B)) robot commands, *Sequence Command*, for the 3 robots. The robots are gathered in wedge formation and a path is calculated for the squad. The water and border obstacles are avoided both during squad path calculation and individual robot navigation. Results in Table 3 show the performance of the different architectural components between the assignment of the strategic order by the player and sending the separate commands to the robots. As expected the main bottleneck is the Swarm Intelligence where the path finding for each robot in the squad is calculated. While the average execution time ($\overline{x}$) out of 20 measurements is 570 ms, the standard deviation ($\sigma$) is quite large (109 ms) due to the large variation displayed by the arbitration procedure of the subsumption loop checking the different behavioural priorities.

**Table 3.** Swarm Intelligence performance for a player command sent to a squad.

| ms | Player $\rightarrow$ SI | Swarm Intelligence | SI $\rightarrow$ Robot Squad |
|---|---|---|---|
| $\overline{x}$ | 9 | 570 | 7 |
| $\sigma$ | 3 | 109 | 6 |
| Min | 6 | 365 | 2 |
| Max | 15 | 756 | 24 |

Scalability of the Swarm and Robot Intelligence subsumption functionality is measured for a squad of 3 robots moving in line formation. A *Squad* `GoTo(x,y)` command is sent at different intervals ranging from 100 to 5000 ms. The average out of 10 measurements is presented for each interval in Table 4. In order to optimize the path finding algorithm the Swarm Intelligence waits for 300 ms before processing the player commands. This results in longer execution time for short intervals (100 and 250 ms) as more information is computed. Due to the possible recalculations of the Swarm Intelligence and consistency reasons, the Robot Intelligence considers only the last received command for execution by the robot as mentioned in Section 4.4 keeping the processing time stable.

**Table 4.** Scalability of the Swarm and Robot Intelligence subsumption framework.

| Interval(ms) | 100 | 250 | 500 | 750 | 1000 | 1500 | 2000 | 2500 | 5000 |
|---|---|---|---|---|---|---|---|---|---|
| Swarm Intelligence(ms) | 1059 | 740 | 362 | 270 | 375 | 173 | 182 | 314 | 190 |
| Robot Intelligence(ms) | 629 | 696 | 603 | 657 | 500 | 627 | 597 | 587 | 551 |

The average Round Trip Time (RTT) in Table 5 for the physical NXT robots is measured when sending 100 messages between PC to NXT while an overloading thread constantly calculates the product of 2 random 10x10 matrices to simulate other calculations on the robot. Results show delays of up to 2 seconds.

**Table 5.** Round Trip Time from PC to NXT and back in ms.

| (ms) | Number of bytes sent | | | |
|---|---|---|---|---|
| # overloading threads | 140 | 200 | 240 | 300 |
| 5 | 1007 | 1014 | 1293 | 1388 |
| 10 | 1034 | 1162 | 1516 | 1640 |
| 15 | 1176 | 1324 | 1732 | 1898 |
| 20 | 1326 | 1607 | 2042 | 2240 |

## 6 Conclusion

This article presents a swarm-intelligent framework achieving meaningful behaviour at swarm-level, instead of the individual robot-level. Keeping in mind the physical robot squads, the designed framework decomposes strategic player commands into individual robot actions using a layered subsumption architecture. Validation through a 'Capture the Flag' scenario shows that a squad command is split up into several robot commands in a matter of 570 ms.

In the future the robots will be enriched with semantics enabling dynamic discovery of their capabilities. This will result on one hand in run-time planning and assignment of tasks depending on the available robot functionality and on the other in automatic distributed interactions between the robots and their environment enabling seamless communication with devices such as sensors, computers, and cameras.

**Acknowledgments.** A. Hristoskova would like to thank the Special Research Fund of Ghent University (BOF) for financial support through her PhD grant.

## References

1. RoboCup, `http://www.robocup.org/`
2. Gabel, T., Riedmiller, M.: On Progress in RoboCup: The Simulation League Showcase. RoboCup 2010: Robot Soccer World Cup XIV. LNCS, pp. 36–47. Springer, Singapore (2011)
3. Mokarizadeh, S., Grosso, A., Matskin, M., Kungas, P., Haseeb, A.: Applying Semantic Web Service Composition for Action Planning in Multi-robot Systems. In: Fourth International Conference on Internet and Web Applications and Services (ICIW 2009), pp. 370–376. IEEE Press, Italy (2009).
4. Schmickl, T., Thenius, R., Moeslinger, C., Radspieler, G., Kernbach, S., Szymanski, M., Crailsheim, K.: Get in touch: cooperative decision making based on robot-to-robot collisions. Autonomous Agents and Multi-Agent Systems, vol. 18, no. 1, pp. 133–155. Springer (2009)
5. Groß, R., Bonani, M., Mondada, F., Dorigo, M.: Autonomous self-assembly in swarm-bots. IEEE Transactions on Robotics, vol. 22, no. 6, pp. 1115–1130. IEEE Press (2006)
6. Trianni, V., Nolfi, S., Dorigo, M.: Cooperative hole avoidance in a swarm-bot. Robotics and Autonomous Systems, vol. 54, no. 2, pp. 97–103. Elsevier (2006)
7. Nouyan, S., Dorigo, M.: Chain based path formation in swarms of robots. Ant Colony Optimization and Swarm Intelligence, pp. 120–131. Springer (2006)
8. Groß, R., Dorigo, M.: Towards group transport by swarms of robots. International Journal of Bio-Inspired Computation, vol. 1, no. 1–2, pp. 1–13. (2009)
9. Ducatelle, F., Di Caro, G.A., Gambardella, L.M.: Cooperative self-organization in a heterogeneous swarm robotic system. In: 12th annual conference on Genetic and evolutionary computation conference (GECCO), pp. 87–94 (2010)
10. Coltin, B., Liemhetcharat, S., Meriçli, C., Tay, J., Veloso, M.: Multi-Humanoid World Modeling in Standard Platform Robot Soccer. In: 10th IEEE-RAS International Conference on Humanoid Robots, USA (2010)
11. Browning, B., Bruce, J., Bowling, M., Veloso, M.: STP: Skills, tactics, and plays for multi-robot control in adversarial environments. IEEE Journal of Control and Systems Engineering, vol. 219, pp. 33–52. (2005)
12. Dias, M. , Kannan, B., Browning, B., Jones, E., Argall, B., Dias, M., Zinck, M., Veloso, M., Stentz, A.: Sliding autonomy for peer-to-peer human-robot teams. In: 10th Intelligent Conference on Intelligent Autonomous Systems (IAS'08), pp. 332–341. Germany (2008).
13. Turgut, A.E., Çelikkanat, H., Gökçe, F., Şahin, E.: Self-organized flocking in mobile robot swarms. Swarm Intelligence, vol. 2, no. 2, pp. 97–120. Springer (2008).
14. ActiveMQ, `http://activemq.apache.org/`
15. Apache log4j, `http://logging.apache.org/log4j/`.
16. Rampinelli, V.T.L., Brandao, A.S., Martins, F.N., Sarcinelli-Filho, M., Carelli, R.: A multi-layer control scheme for multi-robot formations with obstacle avoidance. In: International Conference on Advanced Robotics (ICAR 2009), pp. 1–6. IEEE Press, Germany (2009)
17. Gamage, G.W., Mann, G., Gosine, R.G.: Formation control of multiple nonholonomic mobile robots via dynamic feedback linearization. In: International Conference on Advanced Robotics (ICAR 2009), pp. 1–6. IEEE Press, Germany (2009)
18. Lego Mindstorms NXT, `http://mindstorms.lego.com/`.