Multi-Agent Based Implementation of an Embedded Image Processing System in FPGA for Precision Agriculture Using UAVs

Érico Nunes, Lucas Behnck, Carlos Eduardo Pereira

Multi-Agent Based Implementation of an Embedded Image Processing System in FPGA for Precision Agriculture using UAVs

Érico Nunes, Lucas Behnck, and Carlos Eduardo Pereira

Federal University of Rio Grande do Sul - UFRGS/ DELET -Av. Osvaldo Aranha, 103 CEP: 90035-190 - Porto Alegre, RS - Brazil
emnunes@inf.ufrgs.br,lucas.pluceno@ufrgs.br,cpereira@ece.ufrgs.br
http://www.ufrgs.br

Abstract. This work proposes a framework and design-space exploration on possible implementations for Multi-Agent based embedded systems for precision agriculture applications using image processing. For this application, we evaluate both purely software-based implementations on different embedded processors, as well as implementations which feature dedicated peripherals for image processing implemented in FPGA. All of the implementations feature agent capabilities provided through the JADE Framework. The proposed Reconfigurable hardware Agent framework features capabilities which allow it to offer high performance for applications such as high resolution image processing. We consider the impact of JADE agent migration to an FPGA platform and evaluate the impact of partially reconfiguring the FPGA in this application. The proposed framework is evaluated in an application of use of UAVs for precision agriculture. A faster execution for the image processing algorithms and detection of points of interest (POI) allows for processing images of higher resolution, which may help the accuracy of POI detection. It may also allow for processing an increased number of images in real-time or improve the autonomy of the UAVs.

1 Introduction

Multi-Agent Systems (MAS) are currently a technology targeted for applications which require flexibility, robustness and reconfigurability [12]. Due to their charateristics, there is interest in application and research about MAS in several fields, such as industrial applications [8], area surveillance [11], Unmanned Aerial Vehicle (UAV) missions [17], security [10], smart grids [18], among others.

While this technology is already finding its way into the industry, there are still a drawbacks that hinder large scale application of Agent concepts in industry, such as lack of development tools and standards, lack of skilled design, engineering and maintenance personnel, and many others [8]. Another issue regarded as a challenge for MAS is real-time control [13]. There have been recent studies in this area as described in [7] and [5].

One common method for implementation of agents is the Java Agent Development Framework (JADE) framework. JADE is a well known Agent framework which simplifies the implementation of MAS through a middleware that complies with the Foundation for Intelligent Physical Agents (FIPA) specifications, among other features [2]. JADE allows the agent to be developed in a high level language (Java), provides management and debugging features and allows easy agent migration.

Another recent topic in Agent development in the implementation of Agents in hardware and communication of hardware and software agents. Some works discuss implementations of MAS with hardware Agents, such as in [1], [15] and [3] Most of the work in this area results in implementation proposals which allow only communication between Agents of the same implementation nature (hardware or software) or allow communication between Agents among different implementation natures only through the use of a custom protocol. Usually, the topic of migrating agents from software to hardware is also out of scope.

The work described in [3] details an architecture proposal for implementing hardware Agents and allowing communication and migration between hardware and software Agents. Hardware agents are composed of an field-programmable gate array (FPGA) to implement the Agent hardware functionality and a host processor to provide the bridge between JADE messages and hardware execution. In this work, a middleware is proposed for managing the FPGA reconfiguration during Agent migration. Through the use of the JADE framework for both implementation natures, it enables hardware agents to communicate through a well known interface and to provide faster Agent execution times.

Some limitations discussed in [3] are that complete FPGA reconfiguration can be costly and that an external host processor has to be provided. This also brings the limitation that only one Agent is considered to exist in the FPGA at a time. FPGA partial reconfiguration was cited as an alternative but was left as future work. Adding FPGA partial reconfiguration to this work is expected to reduce FPGA reconfiguration time and increase flexibility, such as in the number of coexisting hardware agents in a same FPGA.

A further contribution which can be made to [3] is the use of higher performance methods of communication between the host processor and the Agent function in FPGA. The proposed middleware relies on Programmed Input/Output (PIO) register accesses, which becomes suboptimal when transferring larger ammounts of data such as high resolution video or images. Through the use of Direct Memory Access (DMA) communication, for instance, high performance image processing capabilities can be added to the proposed architecture.

This work can be regarded as an extension to [3], and proposes a new framework, through the addition of higher performance methods of communication between the host processor and the FPGA.

The proposed framework is evaluated through a study case regarding the use of UAVs on a precision agriculture application. Aerial images obtained by UAV are processed using a segmentation algorithm which executes the segmentation of crop and soil pixels. The execution time is evaluated on different platforms

through the framework to evaluate the possibility of executing it on an embedded hardware on board the UAV.

This paper is organized as follows: Section 2 presents the proposed framework and discusses the proposed implementation in detail. Section 3 further discusses the applications and challenges of using UAV for precision agriculture. Section 4 presents experimental results for the evaluated implementation options. Section 5 presents conclusions and discusses future works.

2 Reconfigurable hardware Agent framework

This work proposes an architecture in order to allow image processing Agents to be implemented in hardware through the use of an FPGA. The Agent is composed of a host processor which is capable of running the Agent framework, as well as the FPGA.

This architecture is targeted at embedded systems which have reconfigurable hardware capabilities such as an FPGA. The high performance capabilities are targeted to applications which need to transfer large ammounts of data between host processor and FPGA, which is a requirement in applications such as high resolution image processing. Figure 1 shows a detailed view of the architecture inside the FPGA, where the hardware Agent is implemented.

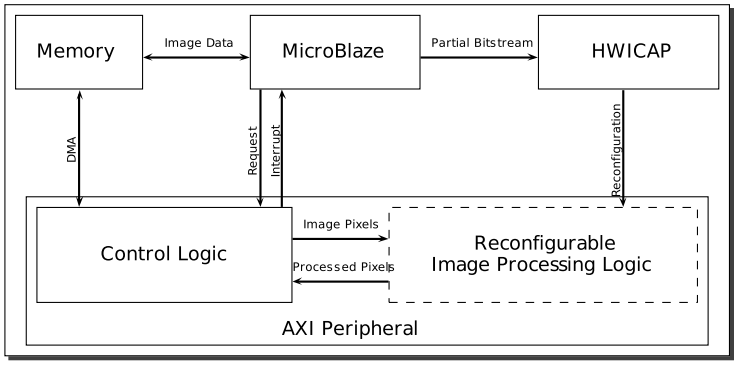


Fig. 1. Hardware components block diagram. Note that besides memory, all of the other components are logic instantiations on the FPGA.

This architecture makes use of a soft core processor inside the FPGA itself. MicroBlaze is a soft core processor provided by Xilinx, which is easily extended with peripherals through the Xilinx Embedded Development Kit (EDK), so it is a suitable option for this architecture. MicroBlaze offers limited processing power for processor intensive algorithms such as image processing, however it can be used for this architecture as the expensive algorithms can be implemented

in FPGA and interfaced easily with it. The use of a lower end processor for this application is also more advantageous as it demands less resources to perform its function, which is to only provide a bridge interface between the framework and the FPGA.

Through the use of the Xilinx EDK, custom peripherals can be added to the MicroBlaze peripheral bus. There are a few different ways to connect custom peripherals to the MicroBlaze, however the recommended way is to add devices to MicroBlaze is through the Advanced eXtensible Interface (AXI) [20]. AXI is part of the Advanced Microcontroller Bus Architecture (AMBA) specification and is currently a standard specification for implementing peripheral devices. These devices can be either AXI4, AXI4-Lite or AXI-Stream, each of which has its own application.

In order to provide high performance for image processing, the architecture features the transfer of images to the FPGA through DMA transfers. An option would be to implement it using PIO (or register-based accesses), but that is very inefficient to transfer data such as an image to the FPGA.

Performance can be maximized if the peripheral is able to read the image from the buffer and write it back without software intervention at all. For example if the peripheral does not interrupt the processor after the whole read/write process. The current implementation uses full AXI with burst capability, in order to provide the best performance.

Figure 2 demonstrates the operations which can be made by the Agent application to the driver in order to perform image processing in FPGA. The first step is to obtain an `mmap` buffer from the driver. The application must use the `mmap`ped buffer because it is allocated by the kernel as a physically contiguous buffer in DMA'able memory. For example, the device may need a physically contiguous buffer it if does not support scatter-gather functionality, and the driver is responsible for ensuring this in behalf of the application. The application must then load the image to be processed to the `mmap`ped buffer. If the image came packed in an image file format and must be unpacked by a file format library, the `mmap`ped buffer can be passed directly to the file format API to avoid an additional memory copy. The application must then request to the driver to initiate the processing step in the device. In this proposal, the same input buffer will be overwritten with the output processed image, in order to reduce the ammount of DMA memory reserved for the application. If the application intends to use the unprocessed image later, it must perform a copy before the processing step. The driver must provide a way to tell the application when the processing is done. This can be implemented as a state which can be polled or through a blocking operation such as a blocking `read` which will only return when the operation is done. When the driver returns that the operation is finished, the application can read the processed image back from the same buffer. After working with the processed image, the application can unmap the memory mapped buffer (`munmap`) and finish.

As also demonstrated by Figure 2, the Agent application is also given the possibility to reconfigure the FPGA by sending an alternate bitstream to a

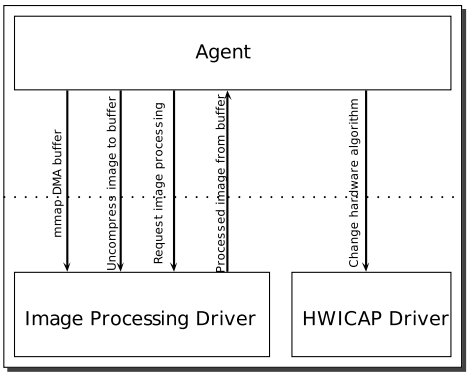


Fig. 2. Operations performed by the Agent application in the driver in order to perform image processing in FPGA.

reconfigurable region, through the Hardware Internal Configuration Access Port (HWICAP) driver.

Another performance boost can be obtained if the peripheral can be designed to match the image encoding format that software uses, so that it eliminates further image manipulation by software later. If the peripheral is able to work with images as they are output by the software encoding/decoding libraries, this makes software implementations much simpler and faster.

If the peripheral supports only contiguous memory buffers for DMA, a large DMA buffer may be required for the image to be read from and written to by the peripheral. Recent Linux kernel versions provide the Contiguous Memory Allocator (CMA) for large buffer allocation. Nevertheless the memory is going to be locked and be unavailable for the rest of the system. Note that this might not be a problem for an application specific embedded system.

An alternative to large DMA buffers is the addition of scatter-gather functionality to the peripheral. Scatter-gather makes use of a descriptor in memory which tells the peripheral the location and size of scattered DMA buffers in memory, allowing the peripheral to use a set of smaller, non-contiguous buffers.

The architecture also includes the possibility of using FPGA partial reconfiguration in order to dynamically change the image processing algorithm in the peripheral. Note that inside the FPGA, there can be as many reconfigurable regions as necessary, but a first thing to note is that there must be a bitstream for each reconfigurable region for each netlist. That is, even if more than one reconfigurable partition is expected to use a same netlist description, each physical reconfigurable partition must have a bitstream specifically generated for it. Each partial reconfiguration bitstream depends on the size of the partition only.

As will be shown in section 4, reconfiguration time for partial bitstreams can be considerably lower than reconfiguring a whole FPGA, so this is another advantage of using partial reconfiguration.

Another benefit of using partial bitstreams is that the static logic only has to be implemented once, and dynamically reconfigurable modules can later be added to the design by only reimplementing the reserved block.

JADE offers the possibility of Agent migration, which can be exploited in this architecture through the use of partial reconfiguration. Initially, JADE can be used in such a way that whenever an Agent migrates to a platform containing a free reconfigurable block, that it reconfigures the block to perform its function in hardware.

3 Case Study: UAV in Precision Agriculture

The case study presented on this paper is a precision agriculture application using UAVs. These UAVs are used as low altitude sensing platforms, used to acquire images from crops in order to identify crops characteristics. The main advantage of such systems is their low cost and easy deployment, compared to other solutions such as satellite based sensing. However, using commercial UAVs on such applications may require new processing units in addition to the existing ones, which are usually responsible for flight control and navigation. These processing units may be part of mission control system, allowing the execution of more sophisticated tasks such as image processing or path planning onboard the UAV [4]. An example of such a mission control module which can be attached to an UAV is presented on Figure 3.

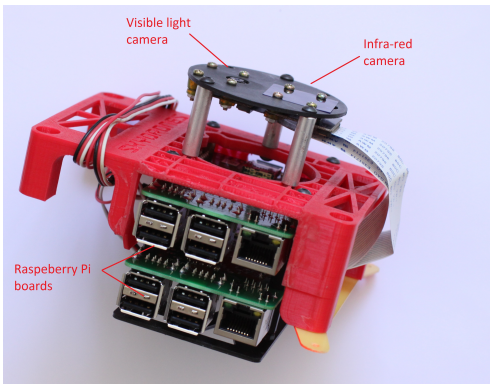


Fig. 3. A sample mission control module composed by two Raspberry Pi computers with visible light and infrared cameras.

Crop images acquired from UAVs may have high spatial and temporal resolution, and each flight mission may yield a large quantity of high resolution images that need to be analyzed in order to obtain useful data [21], such as points of infestation or drought. In order to obtain such information about the crop condition, image processing algorithms are widely employed. However, these

algorithms may demand a large amount of computational power, and executing them on hardware onboard the UAV may impact its energy consumption. The execution time of these algorithms may also be of critical importance if time constraints exist during a given flight mission (for example, a mission where the image processing output is being used as input on the UAV flight path planning). If the execution time of the image processing algorithm is to be considered as the main concern, it is important to note that this sort of algorithm is a typical task that can usually be accelerated when implemented on hardware, and so it is interesting to study if the use of FPGA alongisde the mission control system leads to interesting results.

3.1 UAV and Multi-Agent Systems

Scenarios with multiple UAVs may benefit from MAS implementations. For instance, some authors tackle the coordination of UAVs swarms during search or sensing missions using Agents [14] [6]. In this paper scenario, an Image Processing Agent is studied, which may be onboard an UAV and provide information to other Agents with different roles (such as path planner Agent or another sensing Agent). Some studied features of this Agent are its migration capability and the possibility of it carrying out a partial FPGA reconfiguration, when this Agent is implemented on a platform containing such processing unit. Agent migration may perform an interesting role , where a given UAV may be substituted (due to low battery or failure, for example) and a new UAV may continue to perform its tasks. FPGA reconfiguration, alongside Agent migration, allows the existence of multiple hardware Agents on a single FPGA.

The Agent is implemented using the JADE [2] framework due to its characteristics: the Agent is developed using a high level language (Java), it is a well known framework for Agent development with management and debugging features, and allows easy Agent migration.

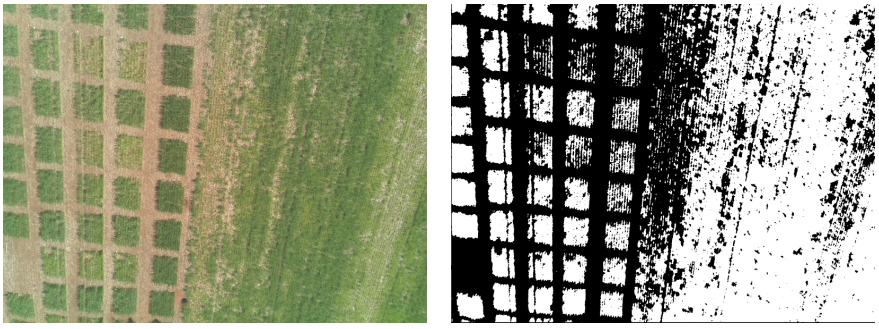
3.2 Image Processing Algorithm

The example algorithm used to evaluate the framework proposed on this work executes the segmentation of a young wheat crop image by discriminating between soil and wheat crop. The first step of the algorithm is the conversion from the RGB colorspace to the excess green index[19], based on equation 1,

$$ex_g = 2g - r - b \tag{1}$$

, where r, g and b are the red, green and blue channels. After the index calculation, a 5x5 Finite Impulse Response (FIR) filter with a homogeneous weight mask is convoluted with the image in order to smooth it. Finally, crop and soil are segmented through the application of a simple threshold.

This simple algorithm could be used to detect low developed wheat plants among crops. The ratio between the total number of “green pixels” and “soil pixels” would be able to give an indicator of the vegetal coverage of the frame, with low ratios indicating possible problems at a given location. Figures 4(a) and 4(b) represent an obtained image and its corresponding post processed version.



(a) Sample crop image. On the left, wheat rectangular wheat patches. On the right, a continuous wheat crop with degraded regions.

(b) Result of the segmentation algorithm.

Fig. 4. Segmentation algorithm example.

4 Results

This section presents and discusses results of a few different implementations of the algorithm described in Section 3. Three target systems were considered: A mainstream Desktop Personal Computer (PC), a Raspberry Pi B+ computer, and a Xilinx ML605 board with a Virtex-6 FPGA. These results were obtained by running the algorithms separately but still not in the JADE environment. All of the experiments were run on the specified target systems in order to obtain the presented results.

A Raspberry Pi B+ computer [16] was considered as a viable processing unit on the case study of this paper, so it is important that its characteristics are presented on this section. A Raspberry Pi is a credit card sized computer with integrated 5MP camera that is suitable for use along UAVs due to its low weight and small dimensions. The model used on this paper is a Raspberry Pi B+ model with an ARM1176JZF-S at 700MHz. A notable feature of the Raspberry Pi for this application is the ARM NEON extension, which provides the processor with Single instruction, multiple data (SIMD) instructions which may accelerate vector operations such as image processing. It is presented as a mission control module on section 3.

The second target is a Xilinx ML605 board, has at its core a Xilinx Virtex-6 FPGA (XC6VLX240T). Inside the FPGA, the board was loaded with a Xilinx MicroBlaze soft core processor. The MicroBlaze version used is 8.40.a from Xilinx EDK 14.2, running at 100MHz. The MicroBlaze processor communicates with its peripherals through an AXI interconnect which also runs at 100MHz. The board provides 512MB of DDR3 memory which is also used by the soft core processor. The soft core processor is configured with a full Memory Management Unit (MMU), which allows it to run Linux. On top of Linux, it runs the JamVM

Java Virtual Machine (JVM) through which it is capable of running JADE. JamVM is an open-source JVM that aims to support the latest version of the Java specification, while at the same time being compact [9].

In order to have a reference for performance evaluation, the algorithm was also tested on a Desktop PC with an Intel Core I5-2450M processor at 2.50 GHz, with Ubuntu 14.04 as operating system.

The image processing algorithm was implemented primarily in the MATLAB environment. The same MATLAB model was used to generate both HDL and C code. In the Raspberry Pi and PC systems, the software version of the code was executed. In the ML605 system, the HDL code was inserted inside an AXI peripheral through the Xilinx EDK and the algorithm was executed in hardware through the FPGA.

Table 1 present the execution time of the image processing algorithm on the different systems. The measured time corresponds to the image processing algorithm execution time, excluding the time used for image capture or image file opening. Three versions of the algorithm with different image resolutions were tested in all of the three targets. Images were tested in resolutions 256x256, 1920x1080 and 2592x1944. The 2592x1944 resolution was chosen as it matches the resolution of the 5MP camera provided by the Raspberry Pi.

Table 1. Execution time for different resolutions of the image processing algorithm, excluding the time used for image capture or image file opening.

Resolution	Processing Time (ms)		
	Desktop PC	Raspberry Pi	ML605 (FPGA)
256x256	4	110	7.99
1920x1080	131	4290	21.87
2592x1944	333	10510	42.01

The software implementations were run as simple Linux applications and therefore may include noise coming from multi-tasking. It should come off as no surprise that the execution in FPGA is faster, but a few more observations can be made about these results.

The Raspberry Pi results show that it is capable of performing image processing relatively fast for low to medium resolution sizes, and would be able to handle a low rate of images to process in real time.

It is notable that the FPGA times are smaller even than the Desktop PC results for the higher resolution, even considering that the FPGA peripheral runs at only 100MHz. It must be considered that the AXI peripheral is capable of performing large bursts to memory and to perform the whole operation in a single cycle after the pipeline is full, in parallel with memory accesses. It is reasonable that the Desktop PC performs better for the small resolution image as it can also provide high performance through SIMD instructions. For the

higher resolution, the software execution may suffer from issues such as cache misses and scheduling in the processor, which may add to the total time.

Table 2. Resource requirements and reconfigurable partition size for different resolutions of the image processing algorithm.

Resource	Available	256x256		1920x1080		2592x1944	
		Required	% Util	Required	% Util	Required	% Util
LUT	12480	1651	14	4367	35	4481	36
FD_LD	24960	711	3	1372	6	1378	6
SLICEL	1680	221	14	586	35	602	36
SLICEM	1440	193	14	506	36	520	37
RAMBFIFO36E1	36	2	6	5	14	10	28

The JADE framework also provides the Agent migration capability, which can be further exploited in the ML605 platform through the use of partial FPGA reconfiguration. For this experiment, the algorithm was implemented in such a way that the Agent application is able to switch the hardware in the reconfigurable region between the three different image resolution implementations of the algorithm. One additional interesting result regarding this is the reconfiguration time required to change this algorithm. As stated in Section 2, the reconfiguration time is only dependent of the reconfigurable partition and partial bitstream size. For a partition size which is roughly three times larger than our larger image processing algorithm, the average reconfiguration block time was measured as 500ms. This was measured on the ML605 board by performing partial reconfiguration on the block described by Table 2, using the HWICAP driver on the board. Table 2 shows the results in resource requirements and partition size.

The similar work presented in [3] shows times in the order of 20s for a full agent migration to hardware including dynamic FPGA configuration, so 500ms can be considered as a reasonable gain. Given that this is a relatively large partition with a large portion of the FPGA reserved for it, it is probably acceptable for Agent applications to perform this reconfiguration on-the-fly. An example application is as demonstrated by this work, where the hardware can be reconfigurated to operate optimally with a given image resolution for image processing.

5 Conclusion

In this work, an architecture proposal for a MAS involving reconfigurable hardware was presented. The architecture was detailed from a practical perspective and it was validated in a case study of an an embedded image processing system for precision agriculture using UAVs. The case study algorithm was also described and experiments implementing the case study algorithm were performed

in three different platforms, of which one is the reconfigurable hardware architecture. The use of partial reconfiguration in this case study was also implemented and evaluated. The proposed framework and experiments are modeled considering the JADE framework. The JADE framework is able to run on all of the presented platforms however it is still not considered in the presented results.

The experiments were performed with both low and high resolution images, and have shown performance gains for the implementation in FPGA even at a relatively low frequency of operation. The use of DMA operations with the AXI bus has shown to be efficient for transferring images to the image processing algorithm implemented in FPGA, and this is a key factor to allow the architecture to maximize performance. The increased performance of the proposed architecture may also allow for processing an increased number of images in real-time or to improve the autonomy of the UAVs.

The proposed architecture is able to communicate with JADE agents through the use of its host processor, which is a soft core processor. The use of the soft core processor has proven to be worthy as it provides enough resources to run the Agent framework and to interface to the FPGA in an easy and efficient way.

Measurements of reconfiguration time were performed and it is noted that reconfiguration time is low even for reconfiguring a significative area of the FPGA. The addition of partial reconfiguration also adds the possibility for multiple hardware agents to coexist independently in the same FPGA platform, which enhances the previous work where the entire FPGA had to be reconfigured. The JADE framework can take advantage of reduced FPGA partial reconfiguration time in order to provide better agent migration from and to hardware.

Further advancements in this work include evaluation of the energy consumption of these implementations in an embedded target. This work has evaluated the performance of the image processing algorithm itself, however end-to-end tests including the JADE framework and actual deployment of the FPGA in the UAV must still be performed.

References

1. Belkacemi, R., Feliachi, A., Choudhry, M., Saymansky, J.: Multi-agent systems hardware development and deployment for smart grid control applications. In: Power and Energy Society General Meeting, 2011 IEEE. pp. 1–8 (July 2011)
2. Bellifemine, F., Poggi, A., Rimassa, G.: Jade–a fipa-compliant agent framework. In: Proceedings of PAAM. vol. 99, p. 33. London (1999)
3. Cemin, D., Gotz, M., Pereira, C.: Reconfigurable agents for heterogeneous wireless sensor networks. In: Computing System Engineering (SBESC), 2012 Brazilian Symposium on. pp. 1–6 (Nov 2012)
4. Doering, D., Benemann, A., Lerm, R., Freitas, E.P., Muller, I., Winter, J.M., Pereira, C.E.: Design and optimization of a heterogeneous platform for multiple uav use in precision agriculture applications. In: World Congress. vol. 19, pp. 12272–12277 (2014)
5. Filgueiras, T., Lung, L.C., de Oliveira Rech, L.: Providing real-time scheduling for mobile agents in the jade platform. In: Object/Component/Service-Oriented Real-

Time Distributed Computing (ISORC), 2012 IEEE 15th International Symposium on. pp. 8–15 (April 2012)
6. Kingston, D., Beard, R.W., Holt, R.S.: Decentralized perimeter surveillance using a team of uavs. IEEE Transactions on Robotics 24(6), 1394–1404 (2008)
7. Krol, D., Nowakowski, F.: Practical performance aspects of using real-time multi-agent platform in complex systems. In: Systems, Man, and Cybernetics (SMC), 2013 IEEE International Conference on. pp. 1121–1126 (Oct 2013)
8. Leitao, P., Marik, V., Vrba, P.: Past, present, and future of industrial agent applications. Industrial Informatics, IEEE Transactions on 9(4), 2360–2372 (Nov 2013)
9. Lougher, R.: Jamvm – a compact java virtual machine. `http://jamvm.sourceforge.net/` (Feb 2015)
10. Megherbi, D., Kim, M., Madera, M.: A study of collaborative distributed multi-goal & multi-agent-based systems for large critical key infrastructures and resources (ckir) dynamic monitoring and surveillance. In: Technologies for Homeland Security (HST), 2013 IEEE International Conference on. pp. 687–692 (Nov 2013)
11. Mustapha, K., Mcheick, H., Mellouli, S.: Modeling and simulation agent-based of natural disaster complex systems. Procedia Computer Science 21(0), 148 – 155 (2013), the 4th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2013) and the 3rd International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH)
12. Pereira, A., Rodrigues, N., Leitao, P.: Deployment of multi-agent systems for industrial applications. In: Emerging Technologies Factory Automation (ETFA), 2012 IEEE 17th Conference on. pp. 1–8 (Sept 2012)
13. Pereira, C.E., Carro, L.: Distributed real-time embedded systems: Recent advances, future trends and their impact on manufacturing plant control. Annual Reviews in Control 31(1), 81 – 92 (2007)
14. Schlecht, J., Altenburg, K., Ahmed, B.M., Nygard, K.E.: Decentralized search by unmanned air vehicles using local communication. In: IC-AI. pp. 757–762 (2003)
15. Schneider, J., Naggatz, M., Spallek, R.: Implementation of architecture concepts for hardware agent systems. In: Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on. pp. 823–828 (Oct 2007)
16. Upton, E., Halfacree, G.: Raspberry Pi user guide. John Wiley & Sons (2014)
17. Ure, N., Chowdhary, G., Toksoz, T., How, J., Vavrina, M., Vian, J.: An automated battery management system to enable persistent missions with multiple aerial vehicles (2014)
18. Vrba, P., Marik, V., Siano, P., Leitao, P., Zhabelova, G., Vyatkin, V., Strasser, T.: A review of agent and service-oriented concepts applied to intelligent energy systems (2014)
19. Woebbecke, D., Meyer, G., Von Bargen, K., Mortensen, D.: Color indices for weed identification under various soil, residue, and lighting conditions. Transactions of the ASAE 38(1), 259–269 (1995)
20. Xilinx, Inc.: EDK concepts, tools, and techniques UG683 (v14.1) (Apr 2012)
21. Zhang, C., Kovacs, J.M.: The application of small unmanned aerial systems for precision agriculture: a review. Precision agriculture 13(6), 693–712 (2012)