SENSORY PROCESSING OPTIMIZATION IN A SMART DEVICE

Jose-Luis Jimenez-Garcia, Jose-Luis Poza-Luján, Eduardo Munera, Juan-Luis Posadas-Yagüe, Raul Simarro
Institute of Control Systems and Industrial Computing
Polytechnic City of Innovation
Polytechnic University of Valencia, Spain
{jojigar1}@inf.upv.es, {jopolu;emunera;jposadas;rausifer}@ai2.upv.es

Abstract

In a distributed control system, where client nodes receive data from distributed sensors in other nodes, it might happen that the same operation about data from a sensor is replicated in different nodes. For that reason, being able to do pre-processing of sensory data in the source node, allows optimizing the client nodes resources. In case of distributing sensory data using services, the sensors that allow that pre-processing are known as smart devices. This article shows a smart device as part of a distributed system that allows to optimize resources in the client nodes. The smart device presented in this is based in a Red, Green, Blue, Depth (RGB-D) sensor. To manage the different acquisitions of the RGB-D it is used a organized processes structure (called plugins) in a topology. The interface of that topology of plugins is named Smart Plugin Topology (SPT) and allows creating plugin combinations, which provides the smart sensor with the ability of doing suitable pre-processing for each client, optimizing the server and local resources of the smart sensor.

Key words: Smart sensor, plugin, built-in system, distributed system.

1 INTRODUCTION

Nowadays, distributed systems based on smart built-in systems are becoming more frequent. They are found in a wide range of systems like: robots, vehicles or video security systems. Each one of them systems uses different sensors in order to complete from easy tasks to complex quests. By a smart sensor, the various functionalities of the sensors may be centralized in order to simplify the client tasks in a distributed system.

Among smart sensors, standing out are Red,Green,Blue (RGB) cameras that have improved in order to be capable of obtaining the depth. These cameras are known as RGB-D. There are various models like xTion [2], Kinect [13] o Senz3D [5], each one of which has specific features: definition, capture distance, price, capture rate and others. Regardless of the choosen model, it is possible of pre-processing the data in order to offer more features to clients. For example, detecting objects with specific color and in a concrete distance. If the distributed system contains more than one client interested in the same data, it is convinient that the smart sensor offers a service to every client interested in that data. In that case, the smart sensor would behave as a smart device.

In this article is introduced a smart device that allows to process and distribute the sensor's data. The smart device shown works by mean of three stages: an initial phase of acquisition, a posterior phase of processing and a final sending phase. In order to complete the intermediate data processing stage, it is necessary to execute various specific processes, called as plugins. What is more, plugins must have capacity of organizing among them, due to the data of one of them being (possibly) relevant to others. For example, a plugin may detect a shape and other plugin might detect color. By composition between both plugins it would be had a third one that would detect objects with the shape of the first plugin and the color of the second one. The structure within the plugins are organized, has been called Smart Plugin Topology SPT. The main objective of the SPT is to manage efficiently the processing stage in order to avoid producing duplicities or an inappropriate use of the system resources. Due to the smart device having SPT, not only the number of devices used is reduced, but it also less computational and energetic cost, and, saves the resources used. Plugins use communication processes to distribute their results. These communications should follow the correspondent requirements of quality of service parameters (QoS), as time, latency, jitter and others.

The article is organized as follows: section 2 contains the information related to implementation. Section 3 shows implementation details of the SPT. Section 4 includes results obtained by the execution of different compound plugins. And last, in section 5, future work of the project and conclusions reached.

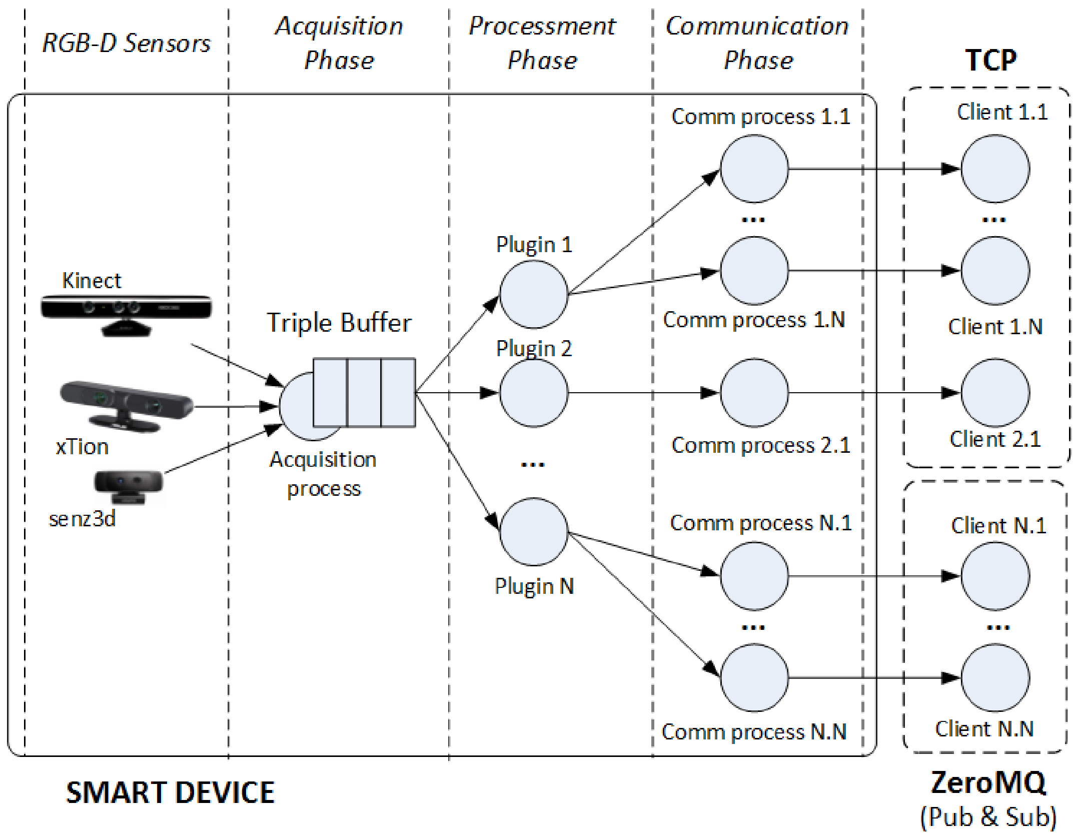

Figure 1: Components and pashes inside the smart device.

2 SMART DEVICE

2.1 SYSTEM DESCRIPTION

The smart device implementation is made by means of the inclusion of an RGB-D sensor, model xTION, for the capture of the pictures, and also a Raspberry PI (RPI) [13] in order to integrate the acquisition, processing and communication stages. To acquire the pictures, it is used the OpenNI v2 [7] API to do the different settings of the device. The acquisition stage is optimized by using a variation of the triple buffer algorithm [9] that allows to improve the obtaining of an only RGB-D sensor when various consumers need the data. In order to make easy the plugins programming, OpenCV v2.4.8 [4] API functions are used. Each plugin sends its result to the consumers, limiting the number of connections that guarantees the QoS of the clients. These consumers might acces to plugins establishing a TCP [12] connection or being subscribers by the use of a Publisher-Subscriber (PUB-SUB) ZeroMQ [8] pattern. These features are shown in Figure 1.

Plugin	Description
Color	Provides RGB image or transformation to BGR.
Gray	Provides the image, but in gray scale.
Contour	Recognizes outlines in the image given by the smart sensor.
Resize	Resizes the image to a specific size.
Soften image	Ereases imprefection like, for example, excesive bright or reflections.
Motion Detection	Detects moving objects.
Color detection	Allows to detect pixels of an specific color.
Gray Depth	Represents a depth matrix in grey scale.
Depth	Distances matrix of the image.
Binary Depth	Provides a binary matrix where 1, represents the existence of an object in the sight range.
Closest Farthest Point	Provides the nearest and furthest object in the sight range.

Table 1: Plugins topology first order

The plugins within Table 1, offer information of the smart sensor with a variable processing level. For example, Color plugin supplies the raw picture or changing the channels, Grey plugin transform from three channels to just one, what entails more processing. Some plugins, like Contour and Motion, need an advanced processing.

2.2 SPT DESCRIPTION

The plugins may be combined between them to generate dynamically new plugins and, then, being able to design specific plugins with the clients requirements. In Figure 2, it is shown the design pattern based in the Composite pattern [10][6] used to join plugins.

The purpose of this pattern consists in making up objects in a tree structure, allowing the different client nodes to deal with first and second order plugins equaly. In this specific case, the first order plugin role is done by different plugins of the SPT and the second order plugin role is the result of merging other first order plugins and producing second order plugins.

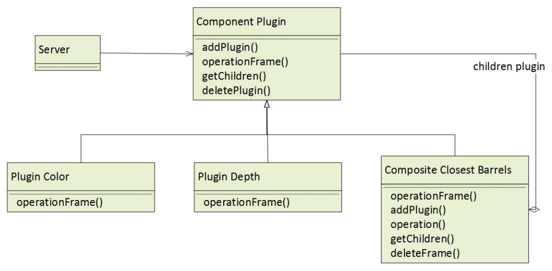

Figure 2: Pattern Composite

The SPT needs a structure that supports the wide plugins variety of the smart device. Besides, the SPT must provide operations that allow using various plugins to join data from the smart sensor and generate enriched information.

3 PLUGIN DESCRIPTION

3.1 COMPONENTS

In this sub-section, it is shown an example of the plugin usage and its organization in the SPT. The example lies in the detection of objects of an specified shape and color, that, besides, is at a certain distance. To do so, various plugins are needed, shown in Figure 3. This case is about a client that is trying to detect circular and red objects that are less than 5 metres far. To attend this petition, the compound of various plugins are needed: one to detect shape (Figure), other to dectect objects at less than 5 meters long (BinaryDepth) and other that detects red objects (ColorDetection). The Figure and ColorDetection plugins need information provided by RGB Plugin, while BinaryDepth plugin needs information provided by Depth plugin. A client requests a second order plugin. The server is using two basic plugins, color and depth. The server establishes a connection with the client and is in charche of creating new plugins in case of not having them (Regardless of the plugin being first or second order). It also will manage intern acquisitions, in order to provide a compound of various plugins. That is, doing pertinent connections between plugins. In the Figure 3 example, the Color and Depth plugins are already created and working. To satisfy the needs of the client, three plugins are needed: Figure, RGB and Depth.

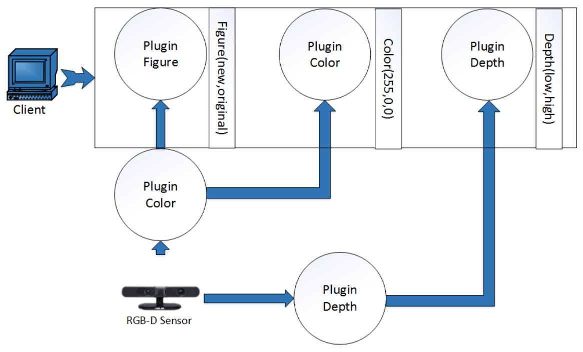

Figure 3: Example of use.

3.2 COMMUNICATIONS

In order to allow the client to inform the smart device about the requirements, a request message is needed. The message header contains information regarding fields size, state and other values. Besides, when a compound plugin petition is done, message header also contains an information addition that can be seen in Figure 4, where it is established the size of the constant concerning to the number of depth plugins family, followed by, and using the same procedure, the color plugins family. This modification causes two header types, a basic one and a larger one, to allow the different plugins representation.

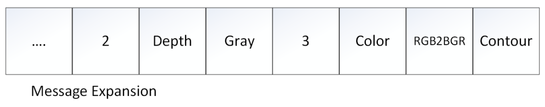

Figure 4: Second order header plugin.

3.3 USING PLUGINS

It is necessary that the plugins will be created dynamically on account of the smart sensor do not know neither when nor what kind of plugin the consumer will require. Two algorithms are executed in the main system thread: Algorithm 1, that creates a plugin, Algorithm 2, that shows the work of a plugin and Algoritm 3, that destroys a plugin.

Algorithm 1, each time a connection regarding the generation of a new plugin is established. In first place it will format the message, distributing the different parts of the information between the specific system variables. One of the most important variables generated by this operation is the buffer id that stores various identifiers of client requested plugins (2). There is one sure thing regardless of the type of the message (basic or extended), the first buffer position will delimit just one plugin, so verifying its existence (3)(13)(14). In case of it not existing, the request type (basic or compound) must be determined using the basic message size (4). Leafs (7) or compounds are generated, the second ones by adding leafs to the inner structure (8)(9). In last place, regardless of the chosen path in the algorithm, the connection must be attached to a plugin (15).

Algorithm 1 Algorithm for creating a Plugin

Require: At least one connection and a valid message.

Ensure: Retrieves the state of the system and waits for a new connection.

```
 1: $Connect() \leftarrow Menssage$
 2: $SplitMessage() \leftarrow id$
 3: if $!Exist(id[0]) \leftarrow boolean$ then
 4:   if $!sizeOf(Mensaje) = 16 \leftarrow boolean$ then
 5:     $CreatePlugin(id[0]) \leftarrow Plugin$
 6:   else
 7:     $CreatePComposite(id[0]) \leftarrow Plugin$
 8:     for $i = 1, sizeOf(id)$ do
 9:       $Plugin.Add(CreatePlugin(id[i]))$
10:     end for
11:   end if
12: else
13:   $Plugin.SearchPlugin(id[0]) \leftarrow Plugin$
14: end if
15: $Plugin.AddConnect()$
```

In the Algorithm 2, a node may request a second order plugin by the extended message format. The compound plugin has an elements list, represented in the algorithm as ListPlugin, and a constant, PDepth, that represents the number of plugins inside the list that are part of the depth processing family. The objective is, by a FDOrigen and a FCOrigen, to apply various processings to make the most of the code and memory, by the use of plugins that might already be in execution serving other nodes. The common calculation of all plugins is getFrame(), which receives the buffer where receive and write information. In lines (1)(2) apply depth pre-processing family and in lines (3)(4) pre-processing of color family. In last place, the operation (7) method does the calculation regarding that compound plugin, that is translated to a final frame, and with the result as desired by the client.

Algorithm 2 Algorithm work of a Plugin

Require: Application plugin with a set order.

Ensure: If the compound plugin not exist it is created according to the algorithm 1.

```
1: for $i = 0, (Size() - PDepth)$ do
2:   $ListPlugin.getPlugin().getFrame(FDDest)$
3: end for
4: for $i = (Size() - PDepth), ListPlugins.size()$ do
5:   $ListPlugin.getPlugin().getFrame(FCDest)$
6: end for
7: $Operation(FDDest, FCDest) \leftarrow Frame$
```

A destruction plugin algorithm method due to a plugin keep consuming resources in a unnecessary manner, because it doesn't have associated connections. The Algorithm 3 is a method that is executed periodically to verify the state of the connections associated to a plugin. The algorithm consists in a nested loop. The outer loop (1) has the task to check all the plugins using the an auxiliar variable (2). The counter variables to use the loops are located on lines (5) and (13). For each plugin returned in the outer loop iteration, the inner loop has the task to check all the connections (4). It is verified on line (6) the conection state. Since a client can unlink to the plugin by itself or conection problems can happen and to lose the conection. If the conection doesn't reach the minimum conection state (6) (7), in example high latency on the conection or the packets don't arrive to the destination, the conection associated to the plugin is deleted. Finally, once all connections associated to the plugin are checked, it is verified if the plugin has still some associated connection. If there aren't connections associated to the plugin, it is eliminated in such a way it doesn't consume extra resources (10) (11).

Algorithm 3 Algorithm Destruccion de un Plugin

```
Require: Exist at least a available plugin.
Ensure: In case the plugin doesn't attend the connections it is eliminated with the algorithm.
1:  for $i = 0, getTallaPlugins()$ do
2:    $auxPlugin = getPlugins() \leftarrow Plugins$
3:    for $j = 0, getTallaConnects$ do
4:      $auxConnect = Plugin.getConnect(i) \leftarrow Connect$
5:      $j+ = 1$
6:      if $!auxConnect.getStatus() \leftarrow bool$ then
7:        $auxConnect.Delete()$
8:      end if
9:    end for
10:   if $Plugin.getTallaConnects() \leftarrow int, 0$ then
11:     $Plugin.Delete()$
12:   end if
13:   $i+ = 1$
14: end for
```

4 CASE STUDY

This section starts with a plugin composition example, to obtain as final result a fusion between a depth matrix and a color matrix, both pre-processed with the format seen. The message used was depth family (depth-gray) and color family (color) so the plugins list size would be three, being the two first the depth family and the last one the color. The composed code is different from the rest compositions by the inner re-definition of the operation method. In this case, the method stills receiving a depth frame and a color frame and stills returning the destination frame, but, internally, it will apply:

$$FrameDest = FCDest * \alpha + FCDest * \beta + \gamma \quad (1)$$

$$FrameDest = saturate(FrameDest) \quad (2)$$

Where FDDest and FCDest are the frames corresponding to the pre-processing application, alpha and beta are constants corresponding to weights of both matrixes and gamma is the scale applied to the fusion. As it can be seen in Figure 5, the result of applying frame fusion with just a few code lines of a method and re-using code by using code from different plugins.

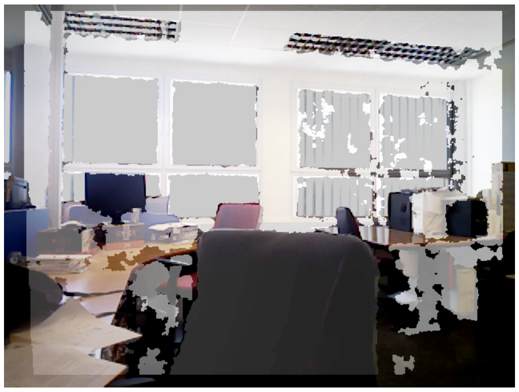

Figure 5: Resultant image from the color and depth plugin fusion.

In Figure 6 can be seen another plugin composition but, in this case, the compound is zeta family (depth) and color family (color - color detection). The result of this composition to follow the red and nearest object and draw its trajectory, the client will receive the result frame of this compound. As it was said, it would be necessary to re-define just the operation method.

$$FrameDest = draw(min(FDDest, FCDest)) \quad (3)$$

In the FDDest frame distances are from the camera and in FCDest are drawn just red objects. Consequently, the min function just have to figure out which is the nearest object by FCDest matrix, when the draw function has calculated the center, it draws a red point and stores it in an array in order to do the sequence. The result is a picture where it can be seen a points sequence that represents the trajectory of that object.

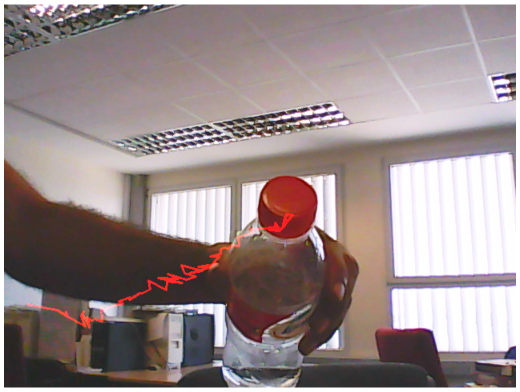

Figure 6: Color Detection.

5 CONCLUSIONS

Finally, as conclusion, in this work the development of a smart sensor aiming to improve and optimize the sensor data flow in a distributed embedded system with limited resources has been detailed. For achieving that goal, is presented the development of a smart sensor endowed with a RGB-D camera as its main sensor device. Next, it has been also described the three main operation phases of any smart resource: acquisition, data process, and sending.

Main contribution of this work is focused on the description of the implemented data processing mechanisms known as SPT which provides resources optimizations and avoids redundant information. The creation, configuration and usage of plugins in a SPT is carefully detailed and its performance has been tested through the presented experiments.

As a future work, it will be studied the benefits and improvements of the addition of smart resources as a part of a real time control system which implements the Control Kernel Middleware (CKM), just as is described in [1]. It also will be evaluated how this devices could enhance the execution of robot oriented CKM tasks just as the behaviour management of the navigation system [11].

Acknowledgment

Coordinated project COBAMI: Mission-based Hierarchical Control. Education and Science Department, Spanish Government. CICYT: MICINN: DPI2011-28507-C02-01/02 and the "Real time distributed control systems" of the Support Program for Research and Development (PAID-06-12) Research Vice-Chancellor of the Polytechnic University of Valencia (SP20120834)

References

[1] Albertos, P., Crespo, A., Simó, J. (2006) "Control kernel", *A key concept in embedded control systems*, In 4th IFAC Symposium on Mechatronic Systems.

[2] ASUS, Xtion PRO. Live. URL [$http://http://www.asus.co.jp/Xtion\_PRO\_LIVE/$].

[3] Borenstein, Greg. Making Things See: 3D vision with Kinect, Processing, Arduino, and MakerBot. " O'Reilly Media, Inc.", 2012. MLA

[4] Bradski, Gary; Kaehler, Adrian. Learning OpenCV: Computer vision with the OpenCV library. " O'Reilly Media, Inc.", 2008.

[5] Creative, creative senz3d. URL [$http://support.creative.com/welcome.aspx$].

[6] Erich Gamma,Richard Helm, Ralph Johnson, Jhon vlissides (2006) "Patrones de Diseño", pp 151-168.

[7] Falahati, Soroush. OpenNI Cookbook. Packt Publishing Ltd, 2013.

[8] Hintjens, Pieter. ZeroMQ: Messaging for Many Applications. O'Reilly Media, Inc., 2013.

[9] Jose-Luis Jimenez-Garcia, Jose-Luis Poza-Luján, Juan-Luis Posadas-Yagüe, David Baselga-Masia, José-Enrique Simó-Ten (2014) "Distribution of Information from a Smart Sensor", *Performance and Results of the Triple Buffering Built-In in a Raspberry PI to Optimize the Distribution of Information from a Smart Sensor*, Distributed Computing and Artificial Intelligence, 11th International Conference, pp 279-286

[10] Judith bishop, (2007) "C# 3.0 Design Patterns", *Design Patterns to Solve Real-World Problems*, pp 49-61.

[11] Munera Sánchez, Eduardo and Muñoz Alcobendas, Manuel and Posadas Yagüe, Juan L and Poza-Luján, Jose-Luis and Blanes Noguera, J Francisco (2014) "Navigation on a Control Kernel", *Integration of Mobile Robot Navigation on a Control Kernel Middleware Based System*, Distributed Computing and Artificial Intelligence, 11th International Conference, pp 477-484

[12] Peterson , Larry L.; Davie, Bruce S. Computer networks: a systems approach. Elsevier, 2007.

[13] Upton, Eben; Halfacree, Gareth. Raspberry Pi user guide. John Wiley and Sons, 2013

[14] Webb, Jarrett; Ashley, James. Beginning Kinect Programming with the Microsoft Kinect SDK. Apress, 2012.