1

A Design Framework for Sensor Integration in Robotic Applications

Dimitrios I. Kosmopoulos
National Centre for Scientific Research "Demokritos"
Institute of Informatics and Telecommunications
153 10 Agia Paraskevi Greece

1. Introduction

The benefits of open robot controllers' architecture have not been ignored by the robot constructors industry and in the recent years the closed proprietary architectures were partially modified to allow access for the integration of external sensors. However, the application integrators are still not able to exploit to the full extend this openness and installations that employ sensors are still the exception in most industrial plants, mainly due to the development difficulties. What is needed to the developers of robotic solutions is a generic method for rapid development and deployment of sensors and robots to enable quick responses to the market needs. This need is mainly dictated by the fact that many such applications sharing common functionality are required in industry. By sharing and reusing design patterns there is a possibility to reduce costs significantly. Therefore, a careful domain analysis that will identify the similarities and differences between the various types of applications and which will provide the infra structure for the development of new ones is of obvious importance.

A major hindrance in the development of sensor-guided robotic applications is the complexity of the domain, which requires knowledge from the disciplines of robot vision, sensor-based control, parallel processing and real time systems. A framework defines how the available scientific knowledge can be used to create industrial systems by embodying the theory of the domain; it clarifies the core principles and methods and the algorithmic alternatives. Therefore new algorithms may be easily integrated through the addition or substitution of modular components.

A design framework should provide the means for the integration of the entailed components and give an overview of their collaboration. The technological advances may then be easily adopted provided that the same cooperation rules will be applied.

In the past many academic and commercial frameworks have been presented, covering parts of the disciplines entailed in sensor-guided robotic applications. However, the available frameworks that can be used for sensor integration in robotic applications were either limited in scope or were not developed targeting to that general goal. From the point of view of the developer of sensor-guided robots the available frameworks treated the general problem in a rather fragmented manner.

This work presents a design framework that defines the basic components that comprise robotic sensor guided systems. It also shows how these components interact to implement the most common industrial requirements and provides some application examples.

Source: Industrial Robotics: Programming, Simulation and Applicationl, ISBN 3-86611-286-6, pp. 702, ARS/plV, Germany, December 2006, Edited by: Low Kin Huat

The rest of this work is structured as follows: the following section defines the context of this research; section 3 summarizes the contribution of this work; section 4 identifies the commonalities and differences between various industrial applications; section 5 describes the user and the design requirements; section 6 describes the basic component, while section 7 shows how these components interact to implement the basic use cases; section 8 demonstrates the validity of the approach by presenting two applications: a sunroof fitting robot and a depalletizing robot. The chapter ends with section 9, which presents the conclusions and the future work.

2. Related work

The literature regarding integration of robotic systems is quite rich. On the one hand there are many middleware systems, which are quite general, treat sensor data processing and the control issues separately and do not address the issue of their interaction in typical use cases. On the other hand some methodologies handle the issue of the cooperation between sensors and actuators but are limited to certain sensor types, control schemes or application-specific tasks. Thus methods that will bridge this gap are needed.

The middleware systems that have been proposed implement the interfaces to the underlying operating system and hardware and they provide programming interfaces that are more familiar to control application builders than the real-time operating system primitives. Some the most interesting of these frameworks are the OSACA (Sperling & Lutz 1996) and the Orca (Orca 2006). The scope of these frameworks is to provide the infra structure and not to specify what the application will functionally be able to do, what data it will accept and generate, and how the end users and the components will interact. They can be used in combination with our framework, which specifies the functionality and the behavior for sensor-guided applications. Some other popular middleware platforms for mobile robots are the Player/Stage/Gazebo (Vaughan et al. 2003), the CARMEN (Montemerlo et al. 2003), the MIRO (Utz et al. 2002), and the CLARAty (Nesnas et al. 2006).

Many other attempts have been made to implement open control objects through the introduction of class or component libraries. Such libraries providing motion control functionality or the related components are the Orca (Orca 2006), the SMART (Anderson 1993), the START (Mazer et al. 1998). The integration of sensors for robot guidance is addressed by libraries such as the Servomatic (Toyama et al. 1996), XVision (Hager & Toyama 1998) but these libraries are limited to certain sensors, control schemes or application-specific tasks thus they are inappropriate for general use.

For object detection in sensory data a large number of frameworks has been presented by researchers and industry. Examples of such frameworks are the Image Understanding Environment (ECV-Net –IUE 2000) and the vxl (VXL 2006). A large number of libraries are available such as the Open CV (Intel 2006), the Halcon (MVTec 2006), etc. Such libraries or frameworks may be used in combination to the proposed methodology.

3. Contribution

The proposed design framework capitalizes on the increased "openness" of modern robot controllers, by extending the capabilities provided by the manufacturer, through intelligent handling and fusion of acquired information. It provides a modular and systematic way in order to encompass different types of sensory input; obviously, this approach can be fully profitable,

when the control system architecture allows for this information exchange to take place. The presented work proposes how to develop software that will undertake the robot control at the end-effector level by profiting of the sensory data. More specifically, it presents:

- A unified approach for implementing sensor guided robot applications considering the restrictions posed by the controller manufacturers and the capabilities of the available control and vision methods.
- A generalization of the available approaches to enable the integration of the popular sensors to any open controller using a wide variety of control schemes.
- Patterns of use including open loop and closed loop control schemes.
- Concrete examples of the development of industrial applications facilitated by the presented framework.

The proposed design is independent of the organizational layer (procedure management, user interface, knowledge base) and execution levels (actuators, sensors) of the application and of the rest layers of the robot controller (integration platform, operating system and hardware).

4. Domain description

A unified approach for the integration of industrial sensors in robotic systems through a common object-oriented framework is reasonable because the necessary abstractions are present for the sensors, for the robotic actuators and for their interactions. This will be clarified through a domain description which will lead to the requirements.

4.1 Sensors

The sensors that are frequently used for industrial robotic tasks may be categorized to contact and non-contact ones. Typical contact sensors that are used in industry are force-torque sensors and tactile sensors. Popular non-contact sensors are cameras, laser sensors, ultrasound sensors which give similar output and inductive, capacitive or Hall sensors for detection of the presence of an object. The processing of the sensory data aims to the identification of features, which in this context normally means the parts of the image with local properties (e.g., points, edges etc.), from which useful information about the environment can be extracted. The use of sensors in most cases presupposes that the inverse sensor model is available, which allows calculation of the environment state from the sensory data.

The sensors can be treated in a unified manner in design due to the fact that the sensory data can be described as vectors or 2-d arrays and what differs is the type of the sensory data (bits, bytes, double precision numbers etc.). Examples of vector images (one-dimensional) are the outputs of the force sensors and the laser sensors; examples of two dimensional images are the outputs of the monochrome cameras, the tactile sensors and the arrays of "binary" sensors; example of image with higher dimensions is the color image, which consists of three monochrome channels (e.g., red, green and blue).

4.2 Robot controllers

The robot controller is the module that determines the robot movement that is the pose, velocity and acceleration of the individual joints or the end-effector. This is performed through motion planning and motion control. Motion planning includes the trajectory definition considering its requirements and restrictions, as well as the manipulator dynamic features. The robot operates in the joint space but the motion is normally programmed in the

Cartesian space due to easier comprehension for humans. Therefore the controller has to solve the inverse kinematic problem (the calculation of the joint states from the end-effector state) to achieve a desired state in the Cartesian space for the end-effector; then the controller must calculate the proper current values to drive the joint motors in order to produce the desired torques in the sampling moments. The industrial robot controllers are designed to provide good solutions to this problem within certain subspaces.
In closed architectures the user is able to program the robot movement by giving the target states and eventually by defining the movement type, e.g., linear, circular etc. The controller then decides how to reach these states by dividing the trajectory into smaller parts defined by interpolation points. The time to perform the movement through a pair of interpolation points is defined by the interpolation frequency; in order to reach the interpolation points closed loop control at the joint level is performed with much higher frequency. Generally the control at the joint level is not open to the programmer of industrial robots.

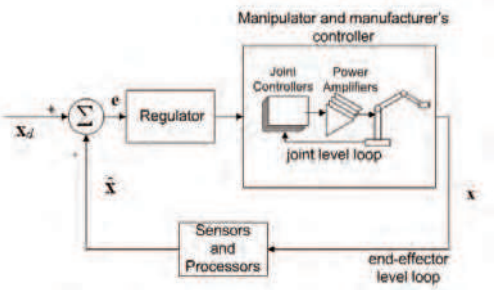


Fig. 1. Closed loop control scheme at the end-effector level. The end–effector's state $\mathbf{x}$ is measured by the sensing devices and the corresponding measurement $\hat{\mathbf{x}}$ is given as feedback; then $\hat{\mathbf{x}}$ is compared to the desired state $\mathbf{x}_d$, the difference is fed to the regulator and the regulation output is sent to the robot controller to move the manipulator. The joint-level loop is also displayed.

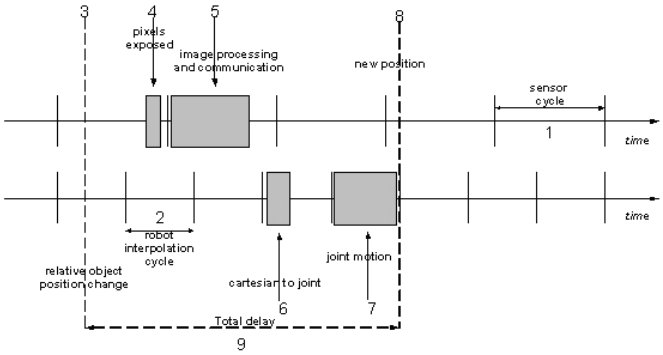


Fig. 2 The cycles of robot and sensor and their synchronization for compensating for a target movement. The total delay from the initial target movement to the new robot pose to compensate for this movement depends mainly on the robot and sensor cycle (adapted from (Coste-Maniere & Turro 1997)).

For tasks requiring low accuracy a "blind" trajectory execution with closed loop control at the joint level is acceptable. However, for the tasks that require high accuracy, closed-loop control at the end-effector level is necessary, during which the next interpolation points have to be calculated in each interpolation cycle using external input; this need is satisfied by the open architecture robot controllers, which unlike the typical closed architecture controllers, permit the user or an external system to define precisely the interpolation points and oblige the end-effector to strictly pass through them. However this restriction may produce oscillations at the end-effector level depending partially on the coarseness of the trajectory. These oscillations may be avoided if an additional regulator is used. The control scheme presented in Fig. 1 is then applied.

4.3 Use of sensors by robots

The sensors can be used for robot guidance at the end-effector level to compensate for the environmental uncertainties. Open and closed loop configurations are then applied. The open loop configurations are the simplest, since the desired robot state is sent to the robot controller and executed by the manipulator without concern about the actual end-effector state. In the closed loop configurations the current end-effector state is compared to the desired one in each interpolation cycle and corrected accordingly in the next cycle.

To identify the commonalities and the differences between the various closed loop control schemes we present the various schemes that may be applied. The schemes identified in (Hutchinson et al. 1996) for visual sensors are generally applicable also to the other sensors types if their type permits it. For the extraction of the desired robot state the "Endpoint Open Loop" scheme assumes perception of only the target's state using non-contact sensors. The "Endpoint Closed Loop" scheme assumes that both the end-effector and target state are perceived and can be applied using contact and non-contact sensors. According to another categorization the closed loop control schemes are divided to the "Position-based", which require extraction of the position for the perceived entities (end-effector and/or target), while the "Image-based" schemes extract the desired robot state by exploiting directly the image features.

In order to implement the aforementioned schemes various methods may be employed according to the application needs. For position-based tasks the physical state of all the observed objects (target, end-effector) is calculated; it is done through optimization methods that use the 3D measurements (or image measurements) in combination to object and sensor models and "fit" the model to the measurements. In image-based systems generally no object or sensor model is used but only an equation that provides the current state from the image measurements, provided that a Jacobian matrix is known (for more details the reader may refer to (Hutchinson et al. 1996)). The Jacobian matrix may be calculated online or (more frequently) offline through a training procedure.

In many applications a single sensor is not adequate for state calculation and in these cases tools such such as Kalman or particle filters may be employed. From the measurement input the system state is calculated, assuming that the system dynamics are linear around the small region where the task is to be executed (for more details the reader may refer to (Valavanis & Saridis 1992)). The filters provide also estimation about the next system state and can also be used for iterative solution of optimization problems in position-based applications.

Another common issue for the closed-loop control systems is that the ideal case in every moment that the robot has to move to reach a desired state this state has to be based on

sensory data that depict the current environmental situation. However, this is usually not possible due to differences in the sensor and robot cycles and due to the time required for processing and transmission of the sensory data (see Fig. 2). Therefore synchronization schemes are necessary that may use prediction for the calculation of the currently required state, based on previous data. For the open loop systems this problem does not exist because the robot moves only after reading and processing the sensory data.

4.4 Application layered architecture

The architecture of a typical robotic application is illustrated in Fig. 3. The left part is adapted from (Valavanis & Saridis 1992), where the levels of organization, processing and execution are presented where intelligence increases from bottom to top. The organization level includes the procedure manager that defines the robotic tasks execution sequence (usually by using a robot programming language), the knowledge base that includes the system's knowledge about the environment and the user interface. The processing level includes the controller, which defines the robot motion by using the processed input from internal and external sensors. The execution level includes the actuators, which perform the requested tasks and the sensors that provide raw sensory data to the system. The right part in Fig. 3 presents the layered architecture of the open robot controllers as presented in OSACA (Sperling & Lutz 1996). The upper layers use the services provided by the lower ones. The hardware layer includes the processing units that are used for the software execution. The operating system controls and synchronizes the use of the resources and acts as the intermediate layer between the hardware and the application software. The communication implements the information exchange between the subsystems. The configuration allows the installation and parameterization of the desired subsystems. The control objects are the subsystems that provide the continuous processing functionality in the application. They are interfaced to the software and hardware system layers through the software- and hardware - API layer, which should provide plug and play capabilities.

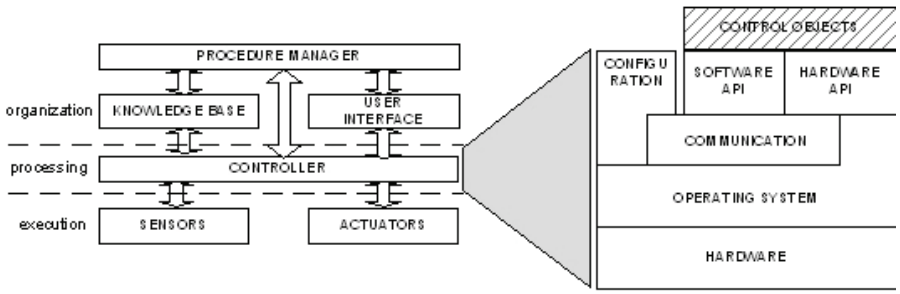


Fig. 3. Typical layered architecture of robotic applications. The organizational level includes the Procedure Manager, the Knowledge Base and the User Interface; the execution level includes the Sensors and the Actuators; the processing level includes the robot controller, which may be analyzed to the open architecture layers.

5. Requirements

5.1 Functional and non functional

A generic framework has to consider some functional and non-functional requirements derived from the description about the sensors, the robot controllers and the control

schemes using sensors. These requirements as seen from the user perspective are briefly presented in the following.
As regards the basic functional requirements, the framework should provide the infra structure and patterns of use for modelling and interfacing of the sensors, should allow sensor integration with industrial robotic controllers in open loop control and closed loop control schemes, should use multiple sensors of the same or different type, should allow object recognition from the sensory data and should use virtual sensory data and manipulators.
The basic non-functional requirements are related to the software quality factors. The composed systems must be characterized by reliability, efficiency, integrity, usability, maintainability, flexibility, testability, reusability, portability and interoperability.

5.2 Use cases

The tasks that the user requires to be executed (use cases) are more or less common for the typical industrial applications and have to be covered by the framework. These are:

- *Manual movement* (offline). The user moves the robot manually using a user interface module by specifying the target state.
- *State reading*. (offline-online). Some sections of the robot trajectory can be executed without strict precision requirements. In these cases the system reads the desired state from the organization level. The consecutive states, from which the robot trajectory is composed, may be defined at organizational level through a robotic language (where high-level tasks are defined) or may be pre-recorded during state teaching.
- *State teaching* (offline). Trajectory sections (set-points) are recorded, while the user moves manually the robot.
- *Parameterization* (offline). During parameterization the data needed for sensor-based control are manually defined and stored. These data are essential for the operation of system and they are retrieved each time a sensor-guided task has to be executed. For each task the system must be parameterized separately. The parameterization may regard the end-effector, the sensor, the regulator and the processing.
- *Training* (offline). Some special system parameters are automatically calculated e.g., inverse Jacobian matrices resulting from feature measurements while the robot executes movement at predetermined steps for each degree of freedom.
- *Sensor-guided movement* (online). The system executes a task in real time. The task includes movement with and without sensory feedback at the end-effector level. When the robot moves without sensory feedback it reads from a database the trajectory that has been stored during teaching. When sensory feedback is required the movement is calculated online using the parameters that have been fixed during parameterization; the parameters are loaded from a database.

5.3 Design requirements and fulfillment

There are several requirements in designing a reusable framework for sensor integration in industrial robotic systems and they are dictated by the user requirements presented in the previous section. The most important design requirements are presented next, followed by the decisions for their fulfillment.
The composing elements have to be modular and decoupled to cover the requirements for reusability, reliability, testability, usability, flexibility, integrity and maintainability. The

semantic decoupling is achieved by grouping functionalities according to general needs and not according to application specific requirements. The interconnection decoupling is achieved by specifying the control objects layer (capsules, described in next section) in a modular way. Each of them has well-specified responsibilities and attributes and communicates with the other ones only through ports using the related protocols (described in next section). By forcing capsules to communicate solely through ports, it is possible to fully de-couple their internal implementations from any direct knowledge about the environment and to make them highly reusable. Virtual sensors and robots can also be used keeping the same protocols. The separation of objects is achieved through the use of the object - oriented C++ language. Interface objects are used wherever possible for component decoupling based on the different roles of the elements.

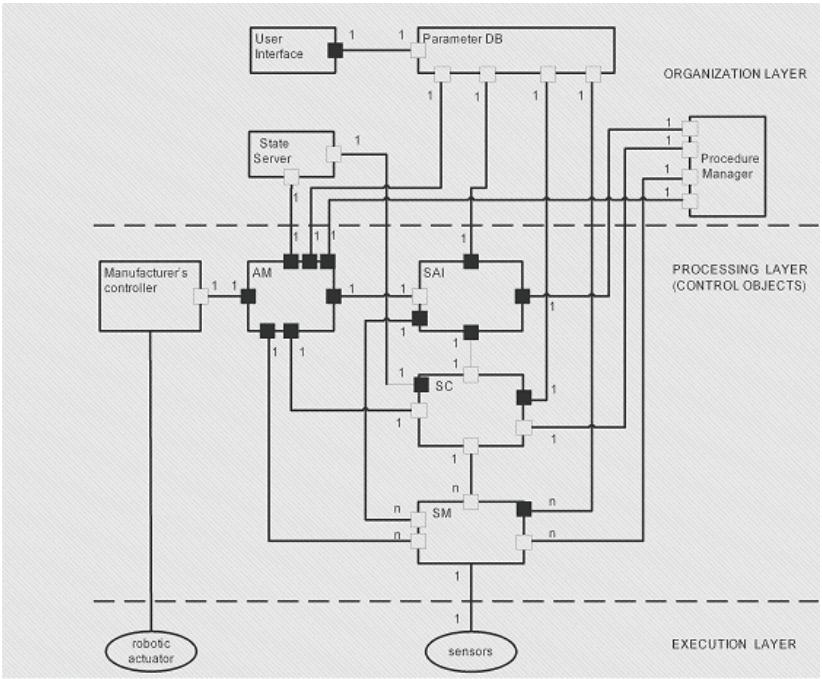


Fig. 4. The decomposition of the processing layer system to control objects (capsules) and their interconnection with modules of the organization and the execution layer (black and white boxes indicate client and server ports correspondingly). It is composed by the *AM* (which communicates with the manufacturer's controller), the *SAI*, the *SC* and the *SM* capsules. The capsules' activation and states are controlled by the *ProcedureManager*. The system parameters are read from the *ParameterDB*, and the predefined states are provided by the *StateServer*. The interface with the user is executed by the user interface, which communicates with the processing layer through the *ParameterDB*.

The framework has to be independent of sensor - specific and industrial robot controllers – specific implementations in order to provide integration of every sensor to any industrial controller type. This is achieved by common communication rules (protocols) for all sensors and controllers.

The framework has to be independent of specific control schemes to ensure wide applicability. It is achieved by defining separate components handling the control scheme – specific aspects of the design (*AM*, *SC* capsules as will be mentioned in next section.
The framework has to be independent of specific vision algorithms and recognition patterns (features) as well. It should allow the organizational layer to change the used algorithms for a task at run time without disturbing the execution at the control object layer. The use of the "strategy" pattern (see e.g., (Douglass 2002)) allows for substitution of algorithms at run-time from higher architectural layers without altering the control flow of the lower layers; it applies to it also facilitates the easy integration of existing algorithms. It also allows easy incorporation of existing code or frameworks. In this manner the fulfillment of the requirements for reliability, efficiency, integrity, usability, maintainability, flexibility, testability, and reusability can be achieved.
The data structures used for handling sensory information are variable mainly due to the type of the acquired data. Therefore the related structures should be flexible enough to incorporate data generated by a wide variety of sensors. The use of class templates that are available in C++ provides a solution to the problem.
A balance has to be achieved between flexibility and usability. The framework should make as less assumptions as possible but on the other hand the less assumptions made the more effort is required to build a system. A good trade-off between flexibility and usability is provided if we build a library of components, which is enriched as long as more applications are implemented.
The implementation of the design decisions at the control object layer is demonstrated in the following subsections. At the class level the design followed does not differ significantly from popular approaches and thus will not be analyzed here.

6. Control objects

This work focuses on the control objects layer, which offers its services independently of the organization, execution and its underlying layers with regard to the open controller architecture. The control objects are usually associated with a particular sensor or actuator type or they can just provide services based on input from other control objects. The sequence of the provided services is defined by the organization level (procedure manager, user interface) and is influenced by the environment.
For the modeling of the system structure we use the UML notation for real-time systems that is proposed in (Selic & Rumbaugh 1998), where the notions of capsules, ports and protocols are defined. Capsules are complex, physical, possibly distributed components that interact with their surroundings through one or more signal-based boundary objects called ports. A port is a physical part of the implementation of a capsule that mediates the interaction of the capsule with the outside world— it is an object that implements a specific interface. Each port of a capsule plays a particular role in a collaboration that the capsule has with other objects. To capture the complex semantics of these interactions, ports are associated with a protocol that defines the valid flow of information (signals) between connected ports of capsules. In the presented design the capsules are the high level system components (control objects), the ports are the interface classes and the protocols are the communication schemes between the components. These are presented in Fig. 4 and explained in the following.

```
1   /* Initialization */
2   {COMMUNICATION WITH DB AND INITIALIZATION OF LOCAL PARAMETERS}
3
4   /*Main loop*/
5   while {CONDITION}
6   {
7      {RECEIVING INPUT}
8      {PROCESSING INPUT}
9      {SENDING OUTPUT}
10  }
11
12  /* Exit */
13  {STORING TO DB AND MEMORY DEALLOCATION}
```

Table 1. Execution scheme of a system process with the initialization phase, the main loop and the exit phase.

Generally the role of the *Controller* that was presented in Fig. 3 is to receive the sensory data from the sensors, to calculate the desired state and communicate the next state to the actuator. It may be divided to the *Manufacturer's Controller* (*MC*), which includes the software and hardware provided by the robot vendor for motion planning and motion control, and the sensor integration modules. These modules are displayed as capsules in Fig. 4 and are the *Actuator Manager* (*AM*), the *Sensor Manager* (*SM*), the *State Calculator* (*SC*) and the *Sensor-Actuator Interface* (*SAI*); their communication with the organization and execution layer modules is also presented.

Before we proceed to the description of the processing-layer capsules it is necessary to describe briefly the organizational level. The role of the *Procedure Manager* is to issue high level commands using events coming from the processing layer. It activates and defines the states of all the processing layer capsules. The *StateServer* is the module of the organization level that maintains the pre-defined robot states, which may include pose, velocity and acceleration. The states may be retrieved from a database or may come as input after online calculation. The *ParameterDB* stores all the parameters and attributes that are useful for the execution of the capsules in the processing layer. The parameters' update is performed online and offline. *ParameterDB* may also act as a link between the *User Interface* and the processing layer.

We begin the description of the processing layer capsules with the *AM*. Its role is to calculate the next robot state and to forward it to the *MC* for execution. The *AM* receives at each robot cycle the current end-effector state from the *MC* and the desired state from the sensory modules and then it calculates the next state (e.g., using a control law - state regulation which helps to avoid the unwanted end-effector oscillations); the next state is then sent to the *MC*. Different control laws can be applied to each DOF. The robot interpolation cycle and the sensor cycle may differ significantly (multi-rate system). Therefore for closed loop control sensor-guided tasks the robot needs to be coupled with the sensors by using the intermediate capsule *SAI*; *SAI* receives asynchronously the desired state from the sensory modules whenever it is made available by *SC*; *SAI* also forwards the current state - that is calculated using sensory input - synchronously to *AM* at each robot interpolation cycle.

The role of the *SM* is to operate at the image level and to extract the image features from the sensory data. On the contrary, the role of the *SC* is to operate at the workspace level and to use the image-level measurements of *SM* to calculate the environmental state (e.g., physical pose) of the end-effector or the target objects. The *SM* sends data requests to the sensors and receives from them the sensory data. After image processing it outputs to the SC the feature measurements. The *SC* receives the feature measurements from the *SM* along with the current actuator state from the *AM* through *SAI*. It outputs to *AM* through the *SAI* the desired state of the end-effector.

Each of the aforementioned capsules operates within a loop in a system process or thread. The scheme of each of those system processes is displayed in Table 1. During initialization the initial local parameters are loaded to memory from the *ParameterDB* and the appropriate memory blocks are initialized (2). The main loop (5-10) performs the processing work. At the beginning of the loop the data from other processes and the parameter database (when appropriate) are read (7). Then the data become processed (8) and the results are sent to the other system capsules or to the organization level (9). The integration platform may implement synchronous or asynchronous communication from either or both sides. The main loop commands may vary according to the capsule state. During the exit procedure, which is executed when the task is finished, the memory is released and the data may be stored into the *ParameterDB*.

The implementation of the design framework uses the services of a communication layer (Fig. 3), which has to be deterministic to cover the needs of a real-time industrial system. The respective communication is performed through the ports presented in Fig. 4.

The communication that is performed within the system follows the producer - consumer concept. A method for communication of modules that reside in different processes is the "proxy" pattern. In this pattern a pair of proxy objects, each of them residing on a different thread/process are used. The local calls to the proxy objects are mapped to inter-process messages (message destinations may be located either at compile time or at run-time through broker objects).

The synchronization protocols are distinguished in synchronous and asynchronous. The "message queuing" pattern can be used for implementing the asynchronous port communication protocol, e.g., through queued messages of unit length. A synchronous protocol can be implemented via the "rendezvous" pattern. The basic behavioral model is that as each capsule becomes ready to rendezvous, it registers with a Rendezvous class and then blocks until the Rendezvous class releases it to run. A detailed description of the use of the proxy, message queuing and rendezvous patterns for inter-process communication and synchronization may be found in (Douglass 2002). In the applications that we created using the proposed framework the ports were implemented as proxy objects.

7. Use cases implementation

In the following we show how to implement the use cases by employing the control objects defined in section 6. At the beginning we briefly describe the tasks that do not require sensory input and namely the *manual movement*, the *state reading* and the *teaching*. Then we describe the general *pre- and post-conditions*. Finally we discuss the interactions for the tasks that require sensory input, which are the *parameterization*, the *training* and the *sensor-guided movement*.

7.1 Operations without sensory feedback

During *manual movement* the user defines through a user interface module the desired robot state, which results in robot movement. Normally the robot manufacturers provide user interface units through which the user is able to move the robot. The user input is then passed to the *StateServer* and the robot reaches the new state after several interpolation cycles calculated in *MC*; for such tasks no third-party software modules are necessary.

During *state reading* the desired trajectory is read from the organizational level and executed without sensory input. The desired states come from the *StateServer*. The state reading task is usually executed before and after tasks that need sensory data; the robot moves without external sensing until a state is reached (e.g., a nominal pose) at which a task that requires sensory input has to be executed e.g., "blind" movement from a "home" robot position to a nominal position where a grasping task is to be executed. The trajectory has been previously recorded during the teaching task described next.

The *teaching* task is performed in order to record a trajectory (interpolation nodes) that will be used later for robot guidance without sensory input (state reading). Its execution has as follows: while the user moves the robot along the desired trajectory each intermediate state is received synchronously from the robot controller and sent to the *StateServer* for storing.

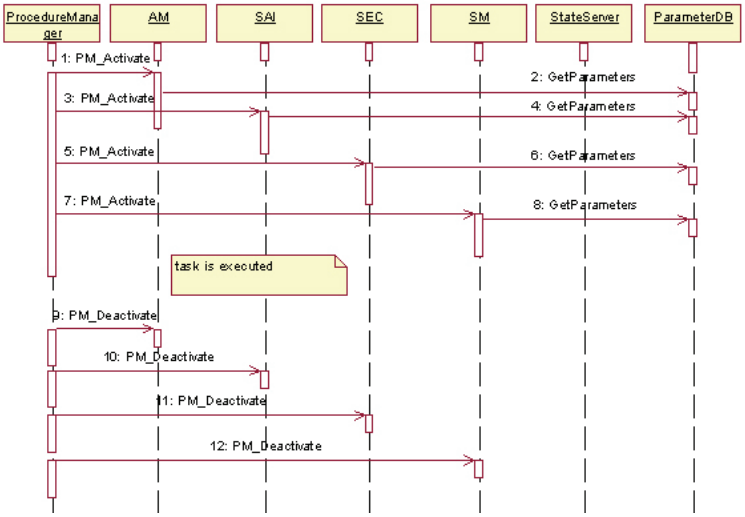


Fig. 5. The general pattern for task sensor-guided movement.

7.2 Pre- and post conditions

We define a common pattern for the conditions that must be satisfied before the invocation of a use case (task) as well as what happens at the end of its execution; this is displayed in Fig. 5.

Before the task invocation the *ProcedureManager* activates the capsules that participate in this specific task and sets their states (1, 3, 5, 7). Through this activation message(s) the ProcedureManager also sends to the capsules the unique id number of the task that is to be executed. This id identifies the type of the task as well the environmental or (and) system status parameters as they were (will be) defined during parameterization or training. Using this id each capsule is able to load the task-specific parameters from the parameter database e.g., threshold values for image processing, control variables for regulation etc. (messages 2,

4, 6, 8). This communication was previously mentioned in line 2 of Table 1. After the task execution the participating capsules are deactivated or eventually set to idle state (messages 9, 10, 11, 12). Before deactivation the new parameters or results are stored into the *ParameterDB* (line 13 of Table 1).
In the tasks that are presented in the following these pre-and post - conditions apply but will omit their presentation for simplicity.

7.3 Parameterization

During the offline parameterization task the user changes manually the parameters used by one or more capsules and is able to evaluate the impact of this change. The parameters are changed to the *ParameterDB* through the user interface and then they are read asynchronously by the active capsules and are applied. If appropriate the parameters may be saved by the user to be used in future tasks. The interactions for the parameterization of an *SM* capsule are given in the following:
The *ProcedureManager* instructs the *SM* to measure (1). The measurement is executed (2) and the results are sent to the *ParameterDB* (3), from where they are accessed (asynchronously) by the User Interface (4). The steps (2-5) are repeated until the user changes an *SM* parameter e.g., a region of interest (6). The new parameters are read by the *SM* (7) and new measurements are performed (8) and written to the *ParameterDB* (9). The new measurements are read asynchronously by the User Interface (10). The steps 7-10 last for a sensor cycle and are performed until the user decides to store permanently the content of the *ParameterDB* (11).

Fig. 6. A parameterization example for the *SM* capsule. The user changes the *SM* parameters and the *SM* reads them through the *ParameterDB* and applies them in processing. Finally the parameters are saved by the user.

7.4 Training

A sub use case of parameterization with particular interest is the training, which is an automated calculation of some sensor-related system parameters. An example of training is

the calculation of a feature Jacobian, which provides the system state from the image features; another example is the calculation of the camera pose relative to the robot end-effector (calibration). In these training tasks the end-effector moves stepwise along a predefined training trajectory for each degree of freedom of the task space. During each training step the sensor subsystem measures the features on the images that are acquired synchronously by the cameras. When the training movement is completed the required parameters are calculated based on the feature measurements; they may be task-specific (e.g., feature Jacobian matrix) or system-specific (e.g., camera pose with respect to robotic hand). Finally the calculated parameters are stored into the database.

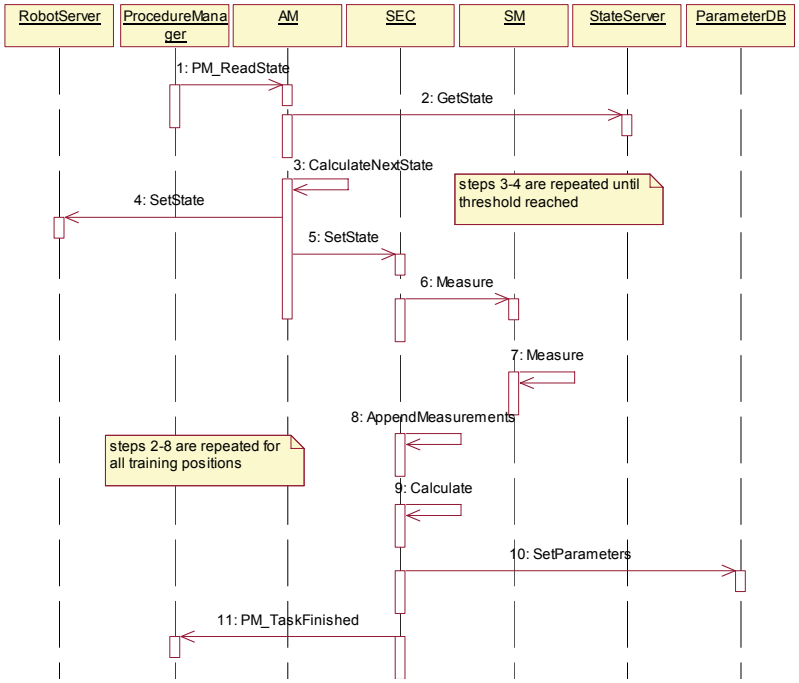


Fig. 7. The capsule interaction during training. When a task requiring sensory input is to be trained the end-effector moves stepwise along a predefined training trajectory for each degree of freedom of the task space and the sensor subsystem measures the image features. When the training movement is completed the parameters are calculated and then stored into the database.

The capsules that participate in training are the *AM*, *SC*, *SM* that communicate with the *StateServer*, *ParameterDB*. The *StateServer* holds the states defining the training movement. The training interaction using a single *SM* instance is presented in detail in Fig. 7 and has as follows:

1. The *ProcedureManager* commands *AM* to read the desired training state from the StateServer.
2. *AM* requests synchronously from *SC* the state vector that defines the next training End Effector State (EES).
3. After receiving the requested state the next EES is calculated using a control law.
4. *AM* sets the next EES to the *RobotServer* to move the robot. The current EES is returned.

Steps 3-4 may be repeated until the distance between the current and the desired training state becomes smaller than a predefined threshold, because high positioning accuracy is required; another scenario is that a timeout occurs without achieving the desired positioning accuracy and in this case we have an exception that either stops training or warns the user. Assuming the first case we have:

5. *AM* sends to the *SC* the currently achieved state, which acts as a trigger to activate the sensors and measure. *AM* waits until the measurement execution.
6. *SC* requests from *SM* to measure and remains blocked waiting for response.
7. *SM* executes the measurement. After measuring, the *SC* receives the measurements and becomes unblocked.
8. *SC* stores the measurement vector. Then the *AM* becomes unblocked.

Interaction steps 2 to 8 are repeated for each training step for all degrees of freedom of the task space. After loop completion:

9. *SC* calculates the required parameters.
10. The parameters are sent to *ParameterDB* where they are stored.
11. The *ProcedureManager* is informed about the task termination.

After step (11) training is finished. The same procedure has to be repeated for all tasks that need training. The steps (2-4) are executed in the robot interpolation cycle, while the duration of step 7 is decided mainly by the duration of the sensor cycle.

7.5 Sensor-guided movement

During sensor-guided movement the robot moves based on the desired states that are calculated after processing sensor input using the pre-defined parameters. This movement may be performed using closed-loop control at the end-effector level, or open loop schemes such as the "look-then-move" or "scan-then-move" that will be presented in the following.

In this task the actuator manager requests from the sensor – actuator interface the measured EES. Due to the difference between the robot and the camera sampling rate the measurement may not be available; in that case a zero state vector is received; else from the state vector (desired state) a new vector is calculated (using a regulator for the reasons explained in section 4.2) and sent to the robot controller. The procedure is repeated until the difference between the desired state and the current becomes very small.

The task execution is performed in real time using the parameters defined during parameterization and training (Fig. 8). The participating subsystems are the *AM*, *SAI*, *SC*, *SM* and the *StateServer*, *ParameterDB*.

The *RobotServer*, *AM* and *StateServer* operate at robot interpolation rate, while the *SC* and *SM*, operate at sensor sampling rate. The *SAI* communicates asynchronously with *AM* and *SC*. Therefore the relative order of the signals between subsystems operating in different rates is just indicative e.g., step 12 is not certain to precede 13. More specifically the interaction has as follows:

1. The *ProcedureManager* commands the *AM* to execute the task using sensory feedback.
2. *AM* requests an EES vector from *SAI*. The *SAI* has no available state error vector, due to the fact that no sensor data have been acquired yet and thus returns a zero vector.
3. *SAI* requests the error from the *SC* asynchronously, triggering the sensor-related part of the system to measure.
4. *AM* calculates from the error and the current state the desired EES using a regulator.
5. *SC* requests a measurement from *SM* synchronously, so that *SC* will be able to calculate the state error.

6. *AM* sends to the robot the correction vector.
7. The call is executed similarly to (2) and zero error is returned since there is no measurement available yet.
8. Similarly to (4) the next EES is calculated using zero error.
9. The next state is set to the robot.
10. Similar to (7) since no measurement available.
11. Similar to (8).
12. Data acquisition and processing are finished and *SM* returns to *SC* the feature measurement vector.
13. Similar to (9).
14. *SC* calculates the EES based on the new measurements.
15. *SC* finished the calculations and calls *SAI* asynchronously and sets the current error vector.
16. *AM* requests from *SAI* the calculated error vector and *SAI* returns it.
17. *AM* calculates the desired EES from the input error using a regulator.
18. *AM* sends to the robot the state calculated in (17), which will move the robot towards the desired EES.

The steps 2-18 are repeated until a threshold is reached. Then:

19. *AM* informs the *ProcedureManager* about the task termination.

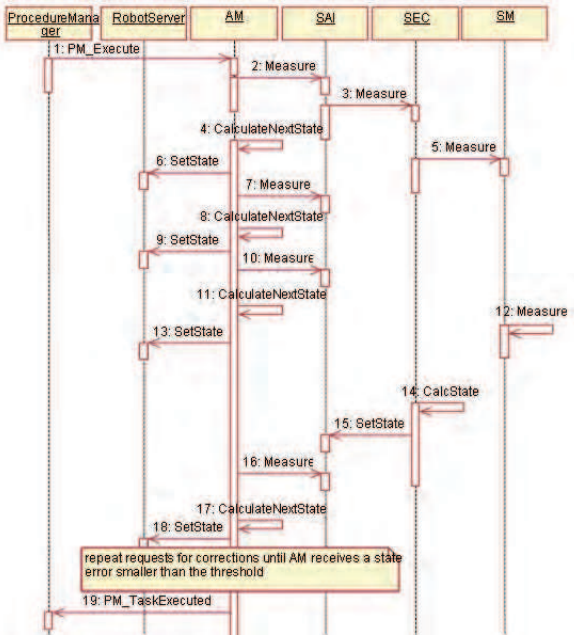


Fig. 8. The capsule interactions during sensor-guided movement for a closed loop system. The actuator manager requests from the *SAI* the measured error of the EES, the *SAI* returns the current state error or zero (due to the different camera and robot interpolation rate) and the regulated pose is sent to the robot controller. The procedure is repeated until the state error becomes very small.

The task of sensor-guided movement in open-loop configurations is much simpler than the respective closed-loop task. We will examine the „look-then-move" and the „scan- then-move" cases. In the former the sensors acquire a local view of the target, while in the latter the sensors acquire a global view of the workspace before executing the task.
In the "look-then-move" scheme, when the robot is at a nominal pose (without moving) the sensors acquire the data, these data are processed, the new state is extracted and then the end-effector moves to reach the required pose. Here there is no need for an *AM* capsule due to the fact that there is no need to apply a regulator at the end-effector level.
In Figure. 9. the general interactions at the control object layer is described. The role of the *StateServer* here is to provide the current end-effector state to the *SC* and to communicate the calculated states based on the sensory data to the rest modules of the organizational level and to the robot controller.
The *StateServer* (after receiving a command at the organizational level) sends to the *SC* the current end-effector state (1) and then the *SC* requests synchronously from the *SM* to acquire sensory data and provide image measurements (2). When the *SM* is ready with data processing (3) the *SC* receives the measurements and continues with state calculation (4). The calculated state is then sent to the *StateServer* (5). Then the state is forwarded to the robot controller.
In the "scan-then-move" approach the robot moves initially to a predefined set of poses, where the sensor is activated and thus a part of the workspace is scanned. The acquired images are processed for feature extraction and registered for each pose using the end-effector pose (or perhaps the images are processed after the execution of the scan movement). From the extracted image features the end-effector states are calculated in order to perform the task.
The *StateServer* (after receiving a command from the execution layer) sends the current state to the *SC*. The *SC* requests synchronously from the *SM* to acquire data and measure (2); when this is done (3) the measurements are registered (4) and the *ProcedureManager* becomes informed about the end of the registration for the current end-effector pose. Steps 1-4 are repeated for all predefined poses; then the next end-effector states are calculated (6) and sent to the *StateServer*. The *StateServer* forwards them appropriately to the robot controller.

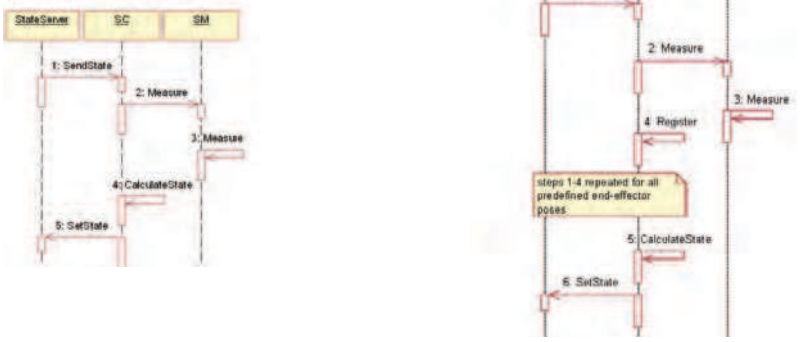


Fig. 9. Sensor-guided movement for a. look-then-move scheme b. the scan-then-move execution scheme.

8. Applications

The applicability of the framework has been verified through the implementation of industrial applications. The UML language has been used for modeling and the C/C++ languages have been used for coding. The first application presented here is the sunroof placement fitting, which aims to fit a sunroof onto a car body on the automobile production line (Kosmopoulos et al. 2002).

We used the 6-DOF manipulator K-125 of KUKA with the KRC-1 controller, which permits the employment of external software for control at the end-effector level. The task of fitting the sunroof on the car body was performed using four CCD cameras, monitoring the four corners of the sunroof opening. The corners were identified as the intersection points of the monitored edges (Fig. 10).

The general scheme introduced in Fig. 4 has been used. For this application four instances of the *SM* capsule have been employed and one of the *AM*, *SAI* and *SC*. As regards the *SM* capsule, the acquired data were of type char due to the monochrome camera images and thus image objects of type unsigned char were defined. In the *SC* capsule we extracted the desired robot state through an image-based, endpoint open loop approach. For the system state calculation we used an extended Kalman filter. The system state was given by the vector:

$$W_k=[x, x', y, y', z, z', a, a', b, b', c, c']^T_k$$

which refers to the end-effector pose error and the corresponding velocities with respect to a nominal state, which is achieved when the sunroof is correctly placed on the target. For the calculation of the end-effector pose we have used an over-determined system through measuring the image coordinates x_i, y_i of the four corner points. The measurement vector $\mathbf{f}_k$ is given by the following equation:

$$\mathbf{f}_k = [x_1, y_1, x_2, y_2, x_3, y_3, x_4, y_4]$$

The system Jacobian has been assumed to be locally constant and has been calculated through training. We have also used classes for coordinate transformations from the target coordinate system to the end-effector, the robot base and other coordintate systems.

For the *AM* capsule we have implemented PID control laws for each degree of freedom. The *SAI* was programmed to send the calculated state to the *AM* as soon it was requested. A zero vector was sent until the next calculated state was available from *SC*.

All the use cases described in section 5.2 have been addressed. The closed loop communication sequences described in sections 7 for a single *SM* capsule in this application were executed in parallel for the four *SM* capsules.

A second industrial application implemented using our generic framework, addresses the depalletizing (or robotic bin picking, or pick and place problem) in the context of which a number of boxes of arbitrary dimensions, texture and type must be automatically located, grasped and transferred from a pallet (a rectangular platform), on which they reside, to a specific point defined by the user (Katsoulas & Kosmopoulos 2006).

We used the KR 15/2 KUKA manipulator, a square vacuum-gripper, and a time-of-flight laser range sensor (model SICK LMS200), which provides 3D points detected by a reflected laser beam. The sensor is mounted on the gripper and the latter is seamlessly attached to the robot's hand (Figure 10).

Due to the fact that only a line is acquired in each sensor cycle, the robot moves the sensor over the pile while acquiring data so that the whole pile can be viewed. The "scan then move" process is utilized for registering one dimensional scan lines obtained by the laser sensor during the robot hand movement, into a 2.5D range image. Finally the object states are extracted from the constructed image and the boxes lying on the upper layer, which is visible to the sensor, are grasped one by one and placed to a new position. For the next layer the same procedure is executed until the pallet is empty.

The general scheme presented in Figure 4 has been implemented omitting the *AM* (due to the fact that no closed-loop control functionality is needed), and the *SAI* (because the robot moves only when the data acquisition is finished and no additional synchronisation is needed); their connections are also omitted.

The communication with the manufacturer's robot controller is achieved through the organizational level (*StateServer*).

The processing layer of the application consists of one instance of the *SM* and the *SC* capsules; the *SM* controls the sensor for data acquisition and the *SC* calculates the state of the target objects, in other words their pose in the world coordinate system (target localization).

Sensor-guided movement follows the "scan then move" pattern, described in figure 9b. Firstly, while the robot is at the initial scan position the *StateServer* sends the current robot state to the *SC* thus triggering a message from *SC* to *SM* to acquire data. The *SM* acquires an image (scan line), which is returned to *SC*. The *SC* receives as input the range image corresponding to the current configuration of the object on the pallet; using the current robot state the coordinates of the image elements are calculated in the world coordinate system. These steps are repeated for all predefined end-effector poses and then the scan lines are combined to a single image. This range image, which holds information about the environment, is processed and from it the target object states (poses) are finally extracted and sent to the organizational level through *StateServer*.

Fig. 10. (a) The sunroof fitting robot using four cameras in a closed loop control scheme and (b), (c) the depalletizing system during scanning and grasping

9. Conclusions and future work

In this chapter we have introduced a design framework for developing software for integrating sensors to robotic applications, provided that open controller architecture is available. The design is general enough and enables the integration of the popular sensors to any open controller using a wide variety of control schemes. By keeping some simple

interfacing rules it is possible to replace modules, e.g., for implementing control laws for the actuator DOFs or for processing the sensor data without much effort. This enables using the freely available or commercial libraries-components maximising the benefit of reusing well – tested modules. It is also possible to integrate multiple instances of the related processes-capsules provided that the interfaces will be the same. The need for a synchronisation module between sensors and actuator in closed loop control schemes has been underlined.

The most common use cases for industrial applications have been presented and the related interaction patterns have been demonstrated. They include some typical open loop and closed loop control schemes. Furthermore, two concrete examples of the development of industrial applications facilitated by the presented framework were demonstrated. Some more applications, not presented here due to confidentiality issues, ranging from robotic measurement to grasping have already employed this approach.

Although the development of design solutions and frameworks for robots is a challenging task we have managed to identify the similarities and differences for the sub-domain of sensor-guided industrial manipulators. The development time for the related applications has been significantly reduced to approximately one third of the initial development time due to reusability of design and components. A good trade-off has been found between over-generalizing and the application-specific design by building concrete examples based on the general principles.

The presented work targets industry-specific systems, where the environment is structured to a significant level. This means that the decisions/use cases taken at the organization level are limited. More complex behaviors e.g., concerning mobile robots moving in unstructured environments are not addressed here.

Anyone who deals with the development of robotic applications can benefit from this work, especially those that seek to couple the mechanical flexibility of industrial robots, with the flexibility to "build" various diverse applications with common characteristics.

In the future the integration in our scheme of a sophisticated configuration tool, such as glue code (code level) and configuration tool in textual or graphical mode (integration and application level), may facilitate the flexibility and the rapid deployment of sensor-specific industrial applications, thus making our framework highly reconfigurable in a dynamic manner. Another objective is to provide functionality through a black box framework from an open integrated environment for the development and testing of the control objects. This environment will support the use of many sensors and actuators and will include their models for simulation purposes employing a variety of vision and control algorithms.

10. References

Anderson, R. J. (1993). SMART: A Modular Architecture for Robots and Teleoperation, *IEEE International Conference on Robotics and Automation*, pp. 416-421, Atlanta, Georgia, May 1993

Corke, P. I. (1993). "Video rate robot visual servoing", In:*Visual Servoing,* K. Hashimoto (Ed.), vol. 7, Robotics and Automated Systems, pp. 257-283, World Scientific, ISBN 9810213646, Singapore

Coste-Manière, E., Turro, N. (1997). The MAESTRO language and its environment: Specification, validation and control of robotic missions. *IEEE International Conference on Intelligent Robots and Systems*, Vol. 2, pp. 836-841, Grenoble, France, September 1997.

Douglass, B.P. (2002). *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*, Addison Wesley, ISBN 0201699567.

Hager, G., Toyama, K. (1998). XVision: A portable Substrate for Real-Time Vision Applications, *Computer Vision and Image Understanding*, Vol. 65, No 1, January 1998, pp.14-26, ISSN 1077-3142

Hutchinson, S., Hager, G., Corke, P. (1996). A tutorial introduction to visual servo control, *IEEE Transactions on Robotics and Automation*, Vol. 12, No 5, May 1996, pp. 651-670, ISSN 0882-4967.

Intel (2006). Open source computer vision library, http://www.intel.com/research/mrl/research/opencv/

Katsoulas, D.K., Kosmopoulos, D.I, (2006). Box-like Superquadric Recovery in Range Images by Fusing Region and Boundary based Information, *Int. Conf. on Pattern Recognition*, Hong Kong, to appear.

Kosmopoulos, D.I, Varvarigou, T.A, Emiris D.M., Kostas, A., (2002). MD-SIR: A methodology for developing sensor-guided industry robots, *Robotics Computer Integrated Manufacturing*, Vol. 18, No 5-6, Oct-Dec 2002, pp. 403-419, ISSN 0736-5845.

Mazer, E., Boismain, G., Bonnet des Tuves, J.M., Douillard, Y., Geoffroy, S., Doubourdieu, J.M., Tounsi M., Verdot, F., (1998). START: An application builder for industrial robotics, *Proceedings of IEEE International Conference on Robotics and Automation*, Vol. 2, pp. 1154-1159, May 1998, Leuven, Belgium.

MVTec (2006). http://www.mvtec.com/

Selic, B., Rumbaugh, J. (1998). "Using UML for modeling complex real-time systems", White Paper.

Sperling, W., Lutz, P., (1996). Enabling open control systems – an introduction to the OSACA system platform, *Robotics and Manufacturing,* Vol. 6., ISRAM 97, New York, ASME Press, ISBN 0-7918-0047-4.

Orca (2006). Orca: components for robotics http://orca-robotics.sourceforge.net/index.html

Toyama, K., Hager, G., Wang, J. (1996). "Servomatic: a modular system for robust positioning using stereo visual servoing", *In Proceedings International Conference on Robotics and Automation*, Vol.3, pp. 2636-2643, April 1996, Minneapolis, USA.

Valavanis, K.P., Saridis, G.N. (1992). *Intelligent Robotics Systems: Theory, Design and Applications*, ISBN 0792392507, Boston.

ECV-Net –IUE (2000). http://www-sop.inria.fr/robotvis/projects/iue/main.html

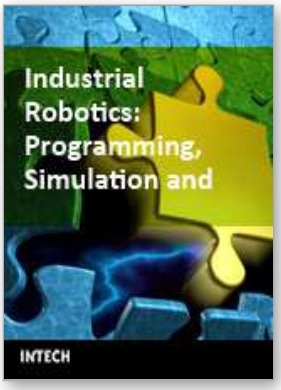
VXL (2006). The vxl homepage, http://vxl.sourceforge.net

Vaughan, R. T., Gerkey, B., and Howard, A. (2003). On device abstractions for portable, reusable robot code, *In Proceedings of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 2421-2427,October 2003,Las Vegas, USA.

Montemerlo, M., N. Roy and S. Thurn (2003). Perspectives on standardization in mobile robot programming: The Carnegie Mellon Navigation (CARMEN) toolkit. *In Proceedings of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 2436-2441, October 2003, Las Vegas, USA.

Utz, H., Sablatnog, S., Enderle, S., and Kraetzschmar, G. (2002). MIRO - middleware for mobile robot applications, *IEEE Transactions on Robotics and Automation*, Vol. 18, No 4, August 2002, pp. 493-497, ISSN 0882-4967.

Nesnas, I. A.D., Simmons, R., Gaines, D., Kunz, C., Diaz-Calderon, A., Estlin, T., Madison, R., Guineau, J., McHenry, M, Hsiang Shu, I., Apfelbaum, D., (2006). CLARAty: Challenges and Steps Toward Reusable Robotic Software, *International Journal of Advanced Robotic Systems*, Vol. 3, No. 1, March 2006, pp. 023-030, ISSN 1729-8806.



Industrial Robotics: Programming, Simulation and Applications

Edited by Low Kin Huat

ISBN 3-86611-286-6

Hard cover, 702 pages

Publisher Pro Literatur Verlag, Germany / ARS, Austria

Published online 01, December, 2006

Published in print edition December, 2006

This book covers a wide range of topics relating to advanced industrial robotics, sensors and automation technologies. Although being highly technical and complex in nature, the papers presented in this book represent some of the latest cutting edge technologies and advancements in industrial robotics technology. This book covers topics such as networking, properties of manipulators, forward and inverse robot arm kinematics, motion path-planning, machine vision and many other practical topics too numerous to list here. The authors and editor of this book wish to inspire people, especially young ones, to get involved with robotic and mechatronic engineering technology and to develop new and exciting practical applications, perhaps using the ideas and concepts presented herein.

InTech Europe

University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China

Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821