Journal of Pattern Recognition Research 1 (2010) 1-23

Robot Localization using Consciousness

Daniela López De Luise and **Gabriel Barrera** *aigroup@palermo.edu*
AIGroup, Universidad de Palermo,
Mario Bravo 1050, Buenos Aires, C1175ABT, Argentina

Stan Franklin *franklin@memphis.edu*
Institute for Intelligent Systems, University of Memphis,
TN 38152, USA

Received September 29, 2010. Received in revised form mmmmmmm dd, yyyy. Accepted mmmmmm dd, yyyy.

Abstract

Autonomous mobile robots require efficient control of their movement. There are several very good approaches for controlling autonomous robots under bound conditions; however, self-adaptation to dynamic environments is very complicated. This paper focuses on part of an autonomous mobile robot prototype named FIC (Fluent Interactive Codelets framework) that has both a main controller and a secondary controller. We conducted an in-depth analysis of the secondary controller implementation. The secondary controller implementation complements the main close-loop controller providing it with further information processing about the current robot status, and suggesting alternate actions. This advisory subsystem implements a new lightweight version of conscious modeling based on the design of Dr. Stan Franklin at the University of Memphis and uses short program pieces named Codelets. This paper focuses on the engineering process performed during FIC prototyping, from requirement collection through design until implementation of the conscious subsystem, as well as how real-world concepts should be properly modeled by Codelet based systems such as FIC.

Keywords: mobile robot, consciousness, design, Codelets

1. Introduction

Mobile robots can be studied in outdoor or indoor environments. This paper describes part of an indoor mobile robot prototype named FIC (Fluent Interactive Codelets framework). It processes several indoor movement parameters based on concept learning technology. The main goal of the module is to provide the robot's close-loop controller with recommendations about which commands should be performed next. Because autonomous mobile robots must navigate in dynamic environments, they require a large collection of environmental information, efficient storage and good processing capabilities. Information processing may require too much computation for real-time implementation. The prototype described here is an autonomous indoor mobile robot adviser with obstacle conceptualization, adapted for dynamic environments. To overcome the loading problem, we propose a decentralized architecture with a dual-loop feedback system: one for immediate decisions (the RTC, or Real Time Controller) and the other for middle to long-term strategies (the RTA, or Robot Task Adviser). The RTC is involved in localization, and aims to determine the position of the robot within its world map. The localization problem can be divided into two steps:

- Process sensor data, to determine a set of possible position coordinates.

- Devise a strategy by which the robot can discover its location.

The RTA is intended for advanced and optional *adaptive* behaviors. It is responsible for the following tasks:

- *Fast Recognition of known objects:* The environment contains several objects which are physically detected and mapped against previously learned concepts. Additionally, they can be grouped into predefined categories such as targetable, forgettable, etc. This way, targetable objects can have unique concepts associated with them and therefore distinct logical treatments.

- *Construction and submission of recommendations to the RTC (FIC 's meta-commands):* FIC commands are designed to be used by the main controller for the following:

    1. *Recognize learned obstacles:* FIC should be able to distinguish between known obstacles and new obstacles physically similar to previously encountered ones. To do this, it is important for the system to define the concept of similarity and to feed this to the localization system along with additional data. This could be defined by the robot's previous experiences or defined ad hoc during setup.
    2. *Define new actions:* The FIC adviser can be trained to detect special situations and to perform low priority commands (akin to suggestions) to the RTC. The data and commands provided by FIC serve as extra information to improve the decisions made by the RTC. However, the time it takes to process this advice may be too long to suit real time mobile requirements, meaning that FIC commands could be discarded or even delayed by the RTC. The robot's Localization System inside the RTC performs a close-loop, with fast reaction to events. It extracts the actual location of the robot in the world map and sends physical movement commands to the effectors. The wider loop, performed by the RTA, is slower and sends optional commands to the RTC. Whenever the controller is performing a high priority activity, (e.g., avoiding an imminent collision) the suggested commands are discarded or considered late. In such situations, commands from the RTC are imposed over commands from the RTA.
    3. *Recognize new obstacles:* There is a set of learned objects, which the system uses to define the set of unknown obstacles because they are not recognized. In these situations, FIC can provide specific advices to the robot.
    4. *Recognize dangerous environments:* This study is focused on a simple indoor mobile robot that is not expected to be exposed to a dangerous environment. However, as the environment becomes more complex (e.g., because of people and other machines moving around), it can also be considered to present greater risk to the robot.

- *Processing and learning based upon experience:* FIC is constantly provided with data of the environment and the robot's location to update its dynamic knowledge database.

This project contributes to existing knowledge in two main areas:

- *Autonomous mobile technology.* The approach improves static world descriptions using movement information and dynamic concept formulations [1] [9] [10] [11]. In this model, obstacles and external references are not necessarily of the same type (i.e., obstacle characteristics), nor have they been physically sensed and mapped in advance.

Using this model, the robot can recognize new obstacles not represented in its previous world map. FIC can also be dynamically targeted to discriminate and recognize similar types of obstacles, such as dangerous obstacles.

- *Concept technology.* The LIDA (Learning Intelligent Distribution Agent) concept model [12] first presented by S. Franklin, and the FLEXO prototype, which is based on Copycat Architecture [6], were used as the starting points to define a new model named ALGOC, which was adapted to FIC restrictions, thus making possible a cognitive robotic system with broader perceptual capabilities as well as the ability to learn.

This project uses part of the original LIDA model and the FLEXO prototype, adding artifacts as components of a meta-model framework named ALGOC. This meta-model was specialized to define a specific model that let a robot move around with a few basic abilities, and some specific interfaces with its controller. In the following sections, this paper presents a general overview of the problem solving strategy that the FIC project is based on (section 2), describes the basic features of the Concept Model (section 3), analyzes the benefits of working with FIC (section 4), and presents an overview of the status of the implementation and future work (section 5).

2. Conceptual Design Strategy for Mobile Robots

This section describes the general problem modeling strategy of the FIC prototype. The approach is described from several perspectives to provide a better understanding of the problem and to propose engineering processes to arrive at a solution.

2.1 Indoor vs. Outdoor Localization Problems

Although a high degree of autonomy is desirable for every mobile robot, it is very difficult to achieve in real world circumstances. In general, an autonomous robot has to be able to:

- Acquire data from its environment
- Interpret that data (turn it into information)
- Perform some task
- Move to a target place
- Avoid certain dangers (collisions, self-destruction, etc.)

There is a set of proposed ways to make a robot autonomous within restricted environments: they are known as indoor mobile strategies and are widely used in factories. They usually determine an object's orientation, position and sometimes type, all in real-time using previous real-world knowledge. Their hardware and software requirements could change depending on whether the robot is on land, underwater, air, underground or in space. Indoor position sensing and navigation began in the 1970s with wired-guidance. Modern robots mostly navigate by sensing natural features. They have the ability to change their paths on the fly to avoid collisions. Most of them navigate within simple and accessible areas, and usually do not perform complex activities, such as climbing stairs.
In contrast to indoor autonomy, there is the outdoor autonomy. It has been achieved mainly in the air, because obstacles are rare. It is difficult to make ground vehicles that are autonomous because they must respond to:

- Three-dimensional terrain

- Great disparities in surface density (i.e., different surfaces have different friction and smoothness)
- Changing weather
- Environmental instability (i.e., obstacles can be removed or change position)
- Energy autonomy considerations (the power supply must be sufficient to enable the robot to perform a set of movements)

To simplify the localization problem and to focus on the adaptability of consciousness to this kind of problem, the FIC prototype has been designed for indoor environments.
Besides defining the complexity of the surroundings, there are extra definitions required for perception of the external world. Exteroception (the ability to perceive external stimuli) can be implemented using a range of sensors (electromagnetic, sound, chemical, temperature, attitude, etc.) and several strategies for processing data. It creates an internal cognitive map of the surrounding world, similar to that defined in [7]: *"...a process composed of a series of psychological transformations by which an individual acquires, codes, stores, recalls, and decodes information about the relative locations and attributes of phenomena in their everyday spatial environment."* This mapping involves further considerations, as it intends to simulate consciousness. As a consequence, two concepts defined by Kitchin are also required: spatial cognition (the cognitive representations of the structure, entities, and relations of space) and its relation to environmental cognition (the awareness, impressions, information, images, and beliefs that people have about environments).

2.2 Robot Controller

Mobile Robot Localization is commonly addressed using laser, ultrasonic, magnetic or other sensory based information. The information from these sensors is used to map the environment and to determine the robot's position using a suitable algorithm. In FIC, it is the duty of the robot's close-loop controller to manage data from external sensors. The current implementation is simple and has very few devices, to focus on the behavior of ALGOC. Future extensions of the prototype will include additional and more sophisticated devices. In the following section, the technologies most commonly used in the field are presented and compared (2.2.1) along with the specific point of view applied to the design of FIC for controllers (2.2.2 and 2.2.3).

2.2.1 Current Controllers

Autonomous robots for indoor environments require an effective and reliable localization technology. Navigation is one of the most important features of mobile robots, of which localization is a key problem. A proper navigation also needs good data from the environment to be able to determine current position. There is a relationship between location and sensory information requirements.
The Table 1 presents a brief comparison of different localization techniques is presented based on specific types of sensors. Generally speaking, it is convenient to complement video and sound information with RF (Radio Frequency) and ultrasonic technologies, such as RFID, ZigBee, wireless devices [14] and Laser Technology [15]. This research study uses ultrasonic technology. RFID technology is reserved for a future implementation that will solve the localization problem more efficiently, including having a higher update rate, better accuracy and scalability at a lower price [16]. The ultrasonic sensors used by FIC receive regulated transmission streams to obtain the distance between the sensor and an obstacle. With that information, FIC updates its world map with the obstacles found.

Table 1: Sensors used for localization.

Sensor	*Range*	*Accuracy*	*Update Date*	*Flexibility*	*Scalability*	*Availability* ✻✻	*Cost*
Ultrasonic	*G*✻	*M*	*H*	*MD*	*G*	*E*	*L*
Laser	*GT*	*G*	*H*	*MD*	*G*	*D*	*VH*
Wireless	*R*	*R*	*M*	*L*	*G*	*MD*	*MD*
Camera	*L*	*L*	*BA*	*L*	*D*	*E*	*L*
Sound	*L*	*L*	*BA*	*L*	*D*	*E*	*L*
RFID	*R*	*M*	*L*	*H*	*G*	*M*	*L*

References: E= Easy, R= Regular, D= Difficult, L= Low, M= Medium, G= Good, GT= Great, H= Hight, VH= Very Hight, MD= Moderate, BA= Depends on the specific Algorithm
✻0<X<100. Depends on the quality of the electronic device. The sensor that we use has a range of a few meters (0 to 5-6 meters).
✻✻How easy the product is to find in an electronic store.

2.2.2 Requirements of a robot Controller

Due to the timing requirements for displacement, FIC has two controllers: one for a fast response to data, and a second one for long term evaluations. The first controller (the RTC) is a typical close-loop, whereas the second (the RTA) is a new type of controller that uses conceptual information. This controller is based on the conceptual model that will be presented in section 3, and is described in more detail in section 4 because it is the main part of the FIC prototype. The rest of this section describes the set of conceptual requirements for the close-loop controller, which could be met by any traditional controller implementation. Although the Monte Carlo Localization (MCL) approach is good due to its ability to model non-linearity and to handle multiple hypotheses, it was not used in the first implementation of the FIC prototype in favor of a simpler approach.

The Monte Carlo method is often used to determine the position of a robot given a map of its environment based on Markov localization. In this method, a large number of hypothetical current configurations are initially randomly scattered in configuration space. With each sensor update, the probability that each hypothetical configuration is correct is updated based on a statistical model of the sensors and Bayes' theorem. All these working elements (current configurations, sensor measurements, and initial world map) must be translated into concepts and/or percepts (special types of concepts that will be presented as part of the Concept Model). This adds further complexity to the initial modeling process.

Similarly, every motion the robot undergoes is applied in a statistical sense to the hypothetical configurations, based on a statistical motion model. When the probability of a hypothetical configuration becomes very low, it is replaced with a new random configuration.

A deep analysis should be performed to evaluate the need to state motion configurations and changes in the conceptual model. Such a study is better performed when the system core had already been implemented and tested, thus reducing the complexity of the first engineering process.

Because MCL requires a previous world model, it is not applicable to unstructured outdoor environments. This problem could be solved by using GPS with a precise receiver, but in urban outdoor locations this is not feasible due to the presence of many buildings, narrow streets, and other obstacles that interfere with satellite signal propagation. Therefore, an alternate approach is required, such as sensor fusion with wheel odometry and inertial data. As mentioned earlier, even though MCL is probably the best choice, its implementation was reserved for a future version of FIC. To reduce complexity, the actual movement algorithm

1-if distance to obstacle is greater than Θ
1.1-move ahead one step
2-else
2.1.if total spin for this movement is 360 degrees (4 obstacles found, no free way to go ahead)
2.2.stop
2.3.go to 4
3-else
3.1-turn right 90 degrees
3.2-go to 1
4.end

Fig. 1: Close-loop algorithm.

is the one presented in Fig. 1.
The only concepts that need to be defined for localization and movement in this algorithm are go-ahead, turn-right, stop, distance, step-distance (meaning distance), spin and obstacle.

2.2.3 Responding to Objects and Obstacles

FIC makes specific use of the ALGOC meta-model to complement the robot controller (referred to here as the close-loop controller). The specific model obtained from ALGOC can be extended to use different sets of sensors. The close-loop controller could also be changed. For the first FIC implementation, we chose the very simple controller shown in the previous section. The second controller was based on the specific model derived from the adaptation of ALGOC to advice properly the first controller.
Because the goal at this stage of the prototype was to test the effects of the ALGOC model on final robot behavior. Thereafter, any object was classified as an obstacle and eluded.
Although it is not one of the goals of this paper, object/obstacle recognition is an interesting problem that the new version of the FIC prototype will need to address. A future implementation will include concepts for managing approximate data about color, temperature, velocity to fire Codelets with the ability to build concepts abstract enough to generalize objects and classifications from previous knowledge. Newly perceived information about the object could reinforce consciousness about surroundings and the proximity of the new objects to the known objects. If the object is new or unclassified, the most similar known object can be used to classify it. Therefore, there will always be a default classification (e.g., consider every new object as an obstacle).
Shape approaches could be used also to improve the results. For instance, if there is a linear border, Codelets could infer that the object is a table or a chair. FIC will generate a workplane for each inference, which would be similar to a blackboard as a work area to work. The preferred workplane will be the one whose percepts contribute to the reinforcement of its concepts.

2.3 ALGOC as part of the FIC prototype

The main contribution of this paper is to derive and implement ALGOC, a new meta-model for robot controllers. It is not just an adaptation of previous conscious theories, but a new modeling framework for consciousness. It does not mimic human brain activities. ALGOC defines a set of basic artifacts to provide assistance and repetition to the modeling process. In this new model, patterns are concepts handled by Codelets with a specific, biased Coderack (a set of small agents each having a very specific portion of code). In autonomous mobile

Table 2: Form for the main steps.

Condition	*Action*	*Object*	Target/Source
condition1	*action*	*Object1*	*type1*

robots, real time answers require fast processing of external information, and critical signals must be processed with a specific workflow that does not exist in original conscious models. This workflow is deeply connected to the regular workflow and the structures it handles. As a consequence of the learning process, the gradual process of reinforcing concepts could be modified for certain situations. In this model, the patterns recognized are not only for external objects but also for events and object classifications.
ALGOC is not just a new implementation of consciousness models. Although it is based on generalized theory (specifically LIDA and FLEXO), it is a framework for developing new specific consciousness, in this case a robot consciousness. The primary, specific differences between the models are described in section 4.3.

3. Modeling Strategy for Robot Problem Engineering

Although it is mainly designed for generating basic responses to stimuli using rapid data collection and localization, FIC is a two-step architecture. Further data processing is performed to implement complex behaviors. The following section describes the engineering steps performed to implements the requirements discussed in previous sections in consciousness model used by FIC. Every step is illustrated with examples taken from the actual design process.

3.1 Outlining Model Requirements

For a good model it is essential to precisely determine the current set of requirements for describing the problem. This is a task that involves collecting and analyzing detailed information. The above sections represent the main part of how this task was performed by the FIC team.

3.2 Determining the Main Steps in the Problem

The problem should now be split into the main steps required to obtain the solution. To find out which steps need to be defined, there is a simple rule of thumb: try to describe the activities using main steps, such as for a recipe, using the form in Table 2.

The first column describes the condition that must be fulfilled to perform the action declared in the second column. Most of the actions are completed unconditionally and therefore this column is usually empty. The third column is the object that is involved in the declared action. The last column simply records whether the action was performed on the object or initiated by it. Table 3 shows the same form, now completed with the steps taken by FIC. It is easy to see whether an object is an external entity feeding the system (as in the case of Data-stream).

There is an important missing step in the ability to declare entities in a concept model: translating the steps being taken to solve the problem into a set of concepts, links and Codelets. To obtain such a set, the Object column should be expanded into a set of concepts and links, while Actions and Conditions should be transformed into Codelets, which is described below.

Table 3: Main steps in FIC.

Condition	*Action*	*Object*	Target/Source
	$Collect$	$Data - Stream$	S
	$Perform$	$Localization$	T
	$Evaluate$	$Collision$	S
	$Perform$	$Simple - Action$	T
	$Evaluate$	$Complementary - Information$	
$Good - Priority$	$Perform$	$Complex - Action$	

Table 4: Initial Version of the Noun and Verb List.

Noun	Verb
$Data - Stream$	Collect
$Localization$	Perform (localization)
$Collision$	Evaluate (collision)
$Simple - Action$	Perform (simple action)
$Complementary - Information$	Evaluate (complementary information)
$Complex - Action$	Perform (complex action)

3.3 Extraction of Noun and Verbs from the Table of Steps

After outlining the main system processes, it is necessary to translate them into a set of basic concepts by parsing the table of steps being performed and extracting the related nouns and verbs. After the initial list is ready, an iterative process extracts more nouns and verbs from the previous list. In this way, the initial set is exploded repeatedly until there are no new nouns and verbs. Table 4 and Table 5 demonstrate how this expansion process is performed for a subset of steps.

After finishing this refinement, the analysis is ready to begin deriving concepts and links. The same procedure is applied to the rest of nouns and verbs in Table 3.

3.4 From Objects to Concepts and Links

Although the derivation process seems straightforward, it usually represents an incomplete version of the final list. The strategy at this step should be rather simple, robust and applicable to any problem that needs to be modeled. Therefore, it should start with lists similar to the ones in Table 5, add more specific information about the activities to be performed to produce a set of better detailed activities (e.g., sub-activities).

3.4.1 General Approach

The modeling process involves the action *get current position*. It has the following tasks:

1. Read odometer value
2. Calculate speed
3. Determine orientation of robot
4. Get previous position in the world-map
5. Deduce new position in the world-map

From this list there is just one more step, translating the previous sentences into simple actions to be assigned as Codelets:

c1. Read position
c2. Read velocity
c3. Calculate new position
...

Translation continues down the list in a similar fashion. However, some of the activities are atomic but others are still too complex to be a Codelet. For instance, c3 could be decomposed as follows:

Table 5: Second Version of the Noun-Verb list.

Noun	Verb
$Data - Stream$	Collect
$odometer - value$	Read odometer value
$obstacle - presence$	Read presence of obstacle
$velocity$	Read velocity
$command$	Read command
$Localization$	Perform (localization)
$World - Map(x, y)$	Evaluate coordinates
$(x, y)Status$	Evaluate next candidate coordinates
$Spin$	Evaluate obstacles

c3.1. Get closeness positions
c3.2. Discriminate obstacles in the neighborhood
c3.3. Evaluate the actual type of command
c3.4. Evaluate continuing the same command
c3.5. Evaluate position changes resulting from each alternate command

The process is repeated to break down any complex function. In this case step c3.1 can be further decomposed as follows:

c3.1.1. Evaluate distance from the current position to every cell
c3.1.2. Select the best threshold for calculating distances within the neighborhood
c3.1.3. Link cells in the neighborhood
c3.1.4. Label links with corresponding distances

This will result in a set of important core activities that can now be referred to as Codelets. However, Codelets act on objects, and only interact with a certain class of objects that represent active concepts. The set of concepts is manipulated to reflect the actual status of conceptualization of the problem (in this case, the problem is a robot and its strategy for moving within the world-map). Certain concepts can be derived from Codelet definitions, whereas others can be extracted from nouns added during problem analysis, as in the example shown in Table 5. Some concepts, called *percepts*, have special status, because they represent direct external information.
Generally speaking, concepts and percepts can be thought of as being similar to traditional programming variables. In the context of Codelet, there are special objects called *links* that describe object status, relationships, and special notations (transient or not). Sometimes they can be treated as traditional working variables, to support the main process.

3.4.2 Information Needed to Apply the General Approach

Returning to the autonomous mobile robot problem, a large amount of specific information is input into the processing mechanisms for modeling Codelets, percepts and links. The main aspects of that information are outlined below:

- *Data Collection.* For simplification, the initial position is not received from sensors, but loaded from a configuration file. In an extended version, a more realistic position (f) will be gathered dynamically from environment. The world map is also uploaded at initiation. In the current version, there is only one ultrasound sensor for obstacle detection that is sensitive up to a distance of 255 cm.

- *Localization.* The list of related nouns and verbs can be easily extracted using the algorithm outlined in Fig. 1.

- *Simple Action (Close-loop).* The localization algorithm involves the following commands: forward, rotation, stop, start.

- *Complex Action (Long-loop).* FIC has a main storage that is named DDB (Dynamic Data Base), that holds key information about life-cycle management and the conceptual model.

3.4.3 Applying the General Approach

Fig. 2 and Fig. 3 show part of the initial result obtained after applying the general approach and specific information provided in previous sections.

As this approach is able to learn, if there is no existing concept for a certain perception, a special Codelet is launched to learn a new obstacle, inserting a new concept and returning a meta-command denoting that the object is unknown in the real world. It also fires a family of explorer Codelets to prescribe a previously defined action (e.g., to avoid an obstacle).

3.5 From Actions and Conditions to Codelets

Figures 4 through 7 show a small subset of the original list of Codelets. Codelets are denoted by *(a)* and are enclosed in triangular brackets. There are also links, denoted by *(l)*.

In the figure, Codelets are shown together with links, that serve as a glue between active Codelets. Some Codelets govern exploring and performing a simple task. Others fire new wavelets that are usually specialized for a certain sub-activity. Whenever a special situation arises, an answer may not be returned, such as, when new obstacles that are slightly different from previously encountered obstacles are detected. Another special situation could result from an ambiguous status or objects. All these situations must be considered in the design and should be encoded as a specific set of concepts that deal with ambiguous statuses and types of non-actions and the links between them, as well as concepts to model status changes and robot reactions.

3.5.1 Testing and Tuning

The design process continues until a good starting set of activities (Codelets), concepts and links are defined. The next step is to start testing. Based on our test results, the lists in Fig. 3 through 7 can be completed to obtain better performance. Thus, the design process becomes iterative and progressive.

4. The Concept Model

This section describes the basic features of the LIDA Concept Model as it was originally stated for modeling problems. Afterwards, a small adaptation named FLEXO is also introduced demonstrate the minimum components required for a real implementation. A comparison between LIDA and FLEXO is provided to demonstrate the effects of any simplification of the original model.

4.1 The Original LIDA Model

LIDA is an extension of IDA (Intelligent Distribution Agent), which is a previous conscious software. IDA is an autonomous software agent that was developed for the US Navy to automate tasks performed by human resource personnel called detailers [13] [17]. IDA was entirely handcrafted with no learning capability. LIDA (Learning IDA) is an extension, that is capable of several forms of learning.

Data Stream (abstract concept)
  Sensory-Data (p)
  Odometer-value (p)
Coordinate (abstract)
  WorldMap (c)
Speed (p)
  Speed (c)
Distance (c)*
Command (c)
  Accelerate (c)
  Deaccelerate (c)
  Turn-right (c)**
  Keep-direction (c)

Notes:
**measure of euclidian distance between two cells.*
***measure of euclidian distance between two cells.*

Fig. 2: Concepts and percepts. Note that *(c)* denotes concept, and *(p)* denotes percept.

Cell (X, Y)
  Position (c)
Status (X, Y)
  Free (c)
  Obstacle (c)
Direction (c)***
    Backward (c)
    East (c)
    Forward (c)
    West (c)
Wold-Map (c)
Threshold-closeness (c)****

Notes:
**** specify general orientation for movement*
*****threshold that states at any time closeness concept*

Fig. 3: Concepts listed according to abstraction level. The leftmost concepts represent higher abstraction.

1.<collect> (a)
  Explores <Sensory-Data> to keep concepts updated and
  fires one or more of the following codelets depending on
  timing criteria.
1.1.read odometer value
  Changed to a more expressive name :
    <evaluate-position> (a)
  Calculates the position of the robot within the <World-Map>.
   It uses the previous position and <odometer-value>.
   At the begining it is calculated from original position loaded
   as a knowledge at time t=0.
1.2.read velocity
  Changed to a more expressive name : <evaluate-speed> (a)
1.3.read command
  Changed to a more expressive name : <next-command> (a)

Fig. 4: Codelets related to external interactions (percepts).

4.1.1 LIDA Overview

The LIDA model is a comprehensive, conceptual and computational model that covers a large portion of human cognition. Based primarily on global workspace theory [18], the LIDA model implements and fleshes out several psychological and neuropsychological theories including situated cognition [18], perceptual symbol systems [19], working memory [4], memory by affordances [20], long-term working memory[21], Slomans cognitive architecture[28] [22], and transient episodic memory [23]. The LIDA computational architecture, derived from the LIDA cognitive model, employs

2.<read-obstacle-presence> (a)
  Activates the <Obstacle-detector> and <Empty-detector> codelets
  according to current <Position>, <Speed> and <World-Map> .
2.1.<Obstacle-detector> (a)
  Searches current <Position> active concepts to detect obstacles.
   If it instantiates an <Obstacle> concept then it also fires
    <Evaluate-Distance> family codelets.
2.2.<Empty-detector> (a)
  Searches current <Position> active concepts to detect if it should
  instantiate a <Free> concept.
2.3.<Empty-Free-Opposite> (l)
  Searches for opposite status.
3.<localize> (a)
  Fires the following position orientation codelets:
3.1.<OpositeEastWest> (l):
  Searches for Opposite relation between cells.
3.2.<OppositeForwardBackward> (l):
  Searches for relationship between cells.

Fig. 5: Codelets and links related to the physical arrangement of obstacles.

4. evaluate-opposite
Changed to a more expressive name :
<search-opposite> (a).
Fires the following codelets
OppositeAccelerateDeccelerate (l)
OppositeRelation (l).
4.1.<OppositeFreeObstacle-detector> (a)
Searches concepts to instantiate
<OppositeFreeObstacle> links.
4.2.<OppositeObstacleFree-detector> (a)
Searches concepts to instantiate
<OppositeObstacleFree> links.
4.3.<OppositeFreeObstacle> (l)
Relates Positions Free that are next to an <Opposite>.
4.4.<OppositeObstacleFree> (l)
Relates Opposite Positions that are next to a <Free>.

Fig. 6: Codelets for evaluating distance.

5.<Evaluate-Distance> (a)
Fires the following codelets:
5.1.<Evaluate-Obstacle-Distance> (a)
Takes <Position> and <World-Map> concepts, calculates and instantiates a <Distance> and <Distance-links> between them for every <Obstacle> active concept.
5.2.<Evaluate-Cell-Distance> (a)
Takes <Position> and <World-Map> concepts, calculates and instantiates a <Distance> for every <Next-Cell> active concept.
5.3.<Relate-foreigner> (a)
Takes <Position> and <World-Map> concepts, linking <Distance> and <Next-Cell> active concepts whenever distance is <Close>.
5.4.<Relate-Neightborhood-Cell> (a)
Takes every <Distance> and links the related <Cell> concepts with less distance (less than <threshold-closeness>).
5.5.<Evaluate-threshold-closeness> (a)
Takes <Distance> concepts and evaluates a new threshold using a weighted distance average.
5.6.<Evaluate-Neightborhood.Cell> (a)
Takes <WorldMap> and calculates distance <Cell>.

Fig. 7: Codelets related to the world map.

several modules that are designed using computational mechanisms drawn from the new Artificial Intelligence (AI) [9] [21].

4.1.2 Cognitive Cycle

The LIDA model and its associated architecture are based on the LIDA cognitive cycle. Every autonomous agent [12], whether human, animal, or artificial, must frequently sample (sense) its environment and select appropriate responses (actions). The agent's life can be viewed as consisting of a continuous sequence of these cognitive cycles. Each cycle constitutes a unit of sensing, processing and acting. Higher-level cognitive processes are composed of many of these cognitive cycles, each called a cognitive *atom*.

The rich inner structure of the LIDA cognitive cycle hypothesized the LIDA model will now be described. More detailed descriptions are available elsewhere [25] [3] [24]. During each cognitive cycle, the LIDA agent first makes the most sense of its current situation by updating its representation of both its internal and external world. Using a competitive process described below, it then decides what portion of the represented situation is most in need of attention. Broadcasting this portion, the current contents of its consciousness, enables the agent to choose and execute an appropriate action. This process is summarized in Fig. 8. The cycle begins when sensory memory senses external and internal stimuli. These low-level features are transmitted to perceptual memory (depicted in the figure, and implemented, as a slip-net [6]) where higher-level features, such as objects, categories, relations, situations, etc. are recognized. These entities comprise the percept that is transmitted to the workspace. It assembles a model of the agents current situation. This percept serves as a cue for two forms of episodic memory: transient and declarative. Each of these forms will answer with a set of events that is locally associated with the cue. In addition to the current percept, the

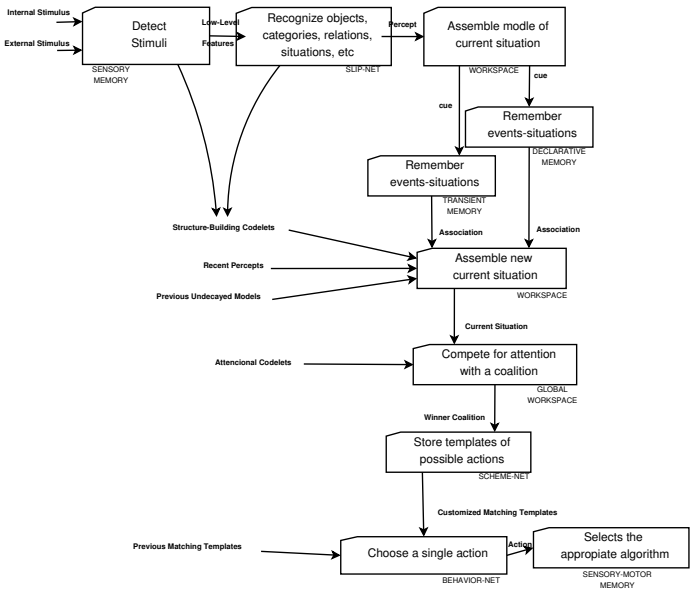

Fig. 8: The LIDA cognitive cycle.

workspace contains recent percepts and the models assembled from them that have not yet decayed. A new model of the agents current situation is assembled from the percepts, the associations, and the undecayed parts of the previous model. The assembly is performed using structure-building Codelets [6]. To fulfill their task, these Codelets may draw upon perceptual memory and even sensory memory to enable recognition of relationships and situations. The newly assembled model constitutes the agents understanding of its current situation within its world. The next phase of the cognitive cycle is to choose which portion of the current situational model to respond to. Alternate portions of the model are formed into coalitions (grouping of model parts) by attention Codelets (specific Codelets that translate information into concepts of the model). One of the coalitions is selected by the agents as the winner. Next, a representation (the most representative part) of the contents of the winner coalition is broadcast globally. The primary recipient is procedural memory, which stores templates of possible actions, including their contexts and possible results. It also stores an activation value for each template that attempts to measure the likelihood that executing a certain action within its context will produce the expected result. The next step is to select a required action. Templates whose contexts sufficiently overlap with the contents of the conscious broadcast can instantiate copies of themselves, with variables that are specific to the current situation. Instantiated templates remaining from previous cycles may also continue to be available. These instantiations are passed to the action selection mechanism, which chooses a single action from one of these instantiations. The chosen action then is transmitted to sensory-motor memory, where it selects the appropriate algorithm and is executed. The action that is taken affects the environment, and the cycle is complete.

4.2 The FLEXO System

FLEXO facilitates the development, testing and optimization of programs using fluid concepts and high-level perception. Because both FLEXO and LIDA are still actively evolving, their reusability and modifiability have been questioned, and a component-based design is being developed to make future adaptations easy. FLEXO should provide all mechanisms that are commonly used in fluid concept implementations and should provide a way to offer the user maximum control. Currently, this project is restricted to the mechanisms used in Copycat/Tabletop. However, FLEXOs architecture should be flexible enough to add the extensions used in newer models (Letter Spirit, Metacat) without too much trouble. The robot administrator should be able to monitor, log and change

almost all of the system's parameters and formulas, at runtime, if possible, and it should be easy to log user-defined parameters and to gather additional statistics and tendencies. In addition, robot administrator should provide components that enable the user to easily enter and alter the details of the domain under investigation:

1. The Code-rack module should be able to load user-defined Codelets, and all its parameters must be easily entered and changed as well. The user (and the Codelets themselves) should be able to call, toggle or clamp each Codelet.

2. A Parameter Manager should offer extensive parameter managing facilities that every parameter should be able to subscribe to, whether they are just values or entire formulas. When designing a new component, the new parameters should be easy to add to this manager. However, it should also be easy for a certain component to retrieve any value it needs.

3. A Log Manager should enable users to log all system (or user-defined) parameters and to create reports over several runs to study the system's behavior.

4. Self-watching modules should be easy to add and use.

The major components of the FLEXO architecture (Fig. 9):

1. *Slipnet.* The Slipnet can be thought of as long-term memory. It is a network of concepts (nodes) connected by conceptual relations (links). However, these nodes carry an ever-changing activation, similar to an electric current, which represents the program's current estimation of the relevance of the concept. When the program deems the concept to be relevant (i.e., when there is an instance of this concept in the given problem), it increases the activation of its node. If the activation passes a certain threshold, the concept has a probability to jump to full activation as the program focuses on it. Over time, this activation decreases gradually as concepts slowly fade away when the program doesn't show renewed interest in a particular concept. Likewise, links have an ever-changing conceptual distance, reflecting the program's current estimation of the closeness of the two concepts connected by a link. There are a limited number of link types, and each type has a descriptive concept (a label), e.g., for example identity or opposite. These labels are treated like any other concept, but they have an extra influence on the conceptual distance of the link type. When a label becomes highly activated, its links shrink as the current conceptual distance between the two concepts decreases, and vice versa. When opposite becomes activated, all opposite concepts in the Slipnet get *closer* to each other because diametrical slippages occur while the program focuses on oppositeness.

2. *Workspace.* The Workspace corresponds to the blackboard in certain programs [26] [22]. It can be likened to a construction site at which an architecture contest is held. Starting at different times, a large number of independent building teams are given the same set of building blocks (i.e., the basic concepts of a problem, e.g., letters) and are challenged to use these to build the most elegant structures, working completely in parallel. The Workspace is a place where perceptual structures are built hierarchically on top of the given input (e.g., three strings of letters).

3. *Coderack.* Each Codelet, when created, is placed in the Coderack, which is a pool of Codelets waiting to run, and is assigned an urgency value (which is a number that will determine its probability of being selected from that pool as the next Codelet to be run. The urgency is a function of the estimated importance of that Codelet's potential action, which in turn reflects the biases embodied by the current state of the Slipnet and the Workspace. Because Codelets are very diverse entities, the only way to offer true flexibility would be to let the user write his or her own Codelets. Although there is dynamic loading, the user may write his or her own Codelets at run-time without having to change (or even know) the underlaying program.

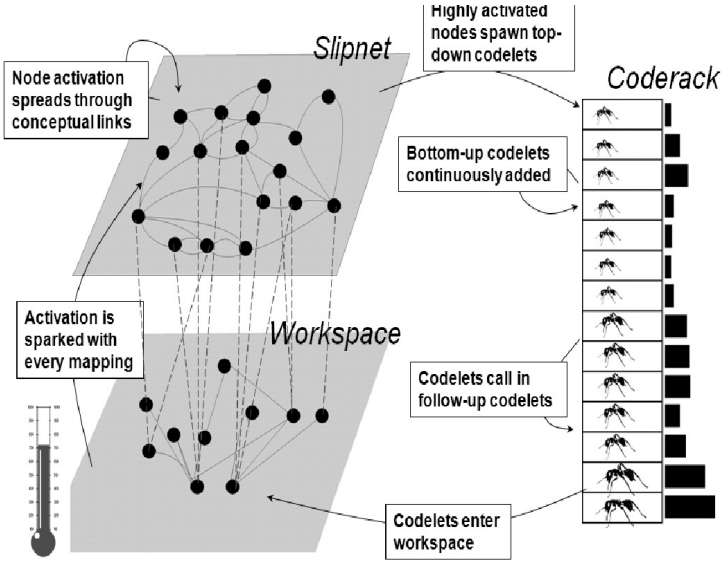

Fig. 9: The general workflow of FLEXO.

The user simply has to define a run-method in a subclass of the Codelet class, compile it, and instruct the program to load it. Access to system parameters is offered through well-defined interfaces. The user can thus concentrate on writing Codelets, with easy access to all of the system's parameters. Internally, the system can streamline the processing by instantiating only one copy of each Codelet.

4.3 LIDA vs FLEXO

Both LIDA and FLEXO are part of the historical evolution of concept modeling. There are two reasons why FIC is based on FLEXO:

- *Dimension:* The first requirement for a robot is to be able to implement conceptual modeling in a lightweight fashion, as it is mandatory to process inputs quickly, and to be able to update close controller loops in a real-time fashion.
- *Simplicity:* The simpler functionality makes FLEXO easy to test and adapt. The first target of the FIC project was to build a working prototype with minimal abilities, just to test the ubiquity of conceptual handling on this kind of problems. The key differences between LIDA and FLEXO will be described in this section to demonstrate the variation between them.

FLEXO lacks of some of the modeling properties required for a good autonomous mobile robot. Therefore, some of the LIDA functionality is applied to FIC. The justification for that is based mainly on the following characteristics of LIDA:

- *Strategy:* LIDA has powerful strategies to accurately model complex concepts. Many aspects of the concept management are represented, including long term memory, learning, and forgetting.
- *Flexibility:* LIDA is flexible enough to accommodate adaptations. In FLEXO no new Slipnet nodes or Codelets can be created. However, LIDA permits new perceptual memory (Slipnet) nodes to be learned. New expectation Codelets, a type of attention Codelet, are spawned with each action selection. Another important advantage is that LIDA can learn new procedures within an extended domain, whereas FLEXO works only in a fixed and predefined domain.

There is a kernel in the prototype that has mostly FLEXO functionality but also retains a part of the LIDA functionality. FIC is not intended to fully re-implement LIDA; however, parts of its

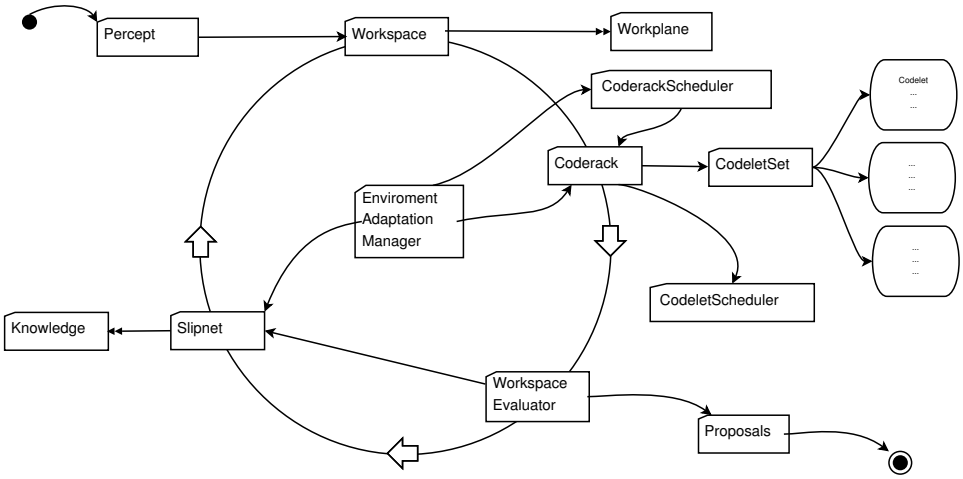

Fig. 10: The general ALGOC workflow.

functionality are included. In addition, there is no simple connection for these two systems. Both systems are intrinsically interconnected. The kernel is a new version of FLEXO redesigned to be more flexible and efficient for real-time purposes with a few LIDA modeling functionalities. It was designed as a flexible framework to be easily extensible for any programmer if there new functionality is required. The FIC architecture and organization are described briefly in the next section.

4.4 ALGOC vs. LIDA and FLEXO

Although ALGOC was inspired by LIDA and FLEXO, there are many key differences. ALGOC is not intended to be a concrete conscious model, but rather a framework for defining conscious models according to specific problems.

Both LIDA and IDA were designed to solve any problem by means of model adaptation. FLEXO goes a step further because it is a specialized working prototype. The key difference is that ALGOC is not a model but a meta-model. There are no Codelets. Deriving a model requires certain classes to exists and be defined with proper design patterns in Java to restrict heritage and composition.

The general life-cycle of ALGOC is shown in Fig. 10. In ALGOC, percepts are managed by the Workspace. At first, Codelets are not fired. Percepts are a specialized subset of concepts that have the ability to be instantly updated at any time, without the need to wait for Codelet processing. This set-up is very different from LIDA, where external stimuli must be initially recognized to define the current situation, and is also different from FLEXO, which just has a Percept Strength to set its relevance.

But there is a second thread of processing that evaluates the current situation using percepts and concepts that is similar to LIDA and FLEXO, which has been included for response to real time problems.

After the Workspace is updated, it builds initial Work-planes with the pure percepts and concepts associated with the activated Codelets.

Codelet activation follows the same principles as it does in LIDA and FLEXO: probability is used to rank Codelets in the Code-rack.

The Codelet Scheduler is similar to the one defined in FLEXO (it fires the most urgent Codelets first). In FLEXO this functionality was not implemented but defined in the theoretical model. This functionality was extended to have distinct behavior for Codelets that just manage percepts.

There is a set of Codelets that get fired in response to Scheduler commands; however the individual Codelets are not handled directly. This constitutes another difference between ALGOC and FLEXO and LIDA: in ALGOC, Codelets are handled in sets. It is expected that any specific model could define sets of the appropriate size, ranging from one to many Codelets. LIDA's life-cycle has something similar, but it is restricted to certain sets: structure building Codelets, recent percepts and

attentional Codelets. In ALGOC, these sets are defined according to the specific type of model. The result is, after the design process described in section 2, there is a classification of Codelets into categories managed by the CodeletSet artifact. Any other task decomposition should be addressed during the design process, and will result in specific relationships joined by links.
There is another Scheduler, the Code-rack Scheduler, which is only present in ALGOC and serves as an interactive tool to allow incremental design in parallel with the working model. That is, if the model detects an unknown concept using a threshold and similarity criteria, the Environment Adaptation Manager (EAM) will detect it and deploy tools to extend the current model.
EAM also goes further and provides a toolbox to manage other aspects of the environment: physical parameters describing sampling, synchronism parameters, multi-threading, Codelet activation acceleration or deceleration parameters for the Codelet Scheduler, initial knowledge treatments, etc.
The last two functionalities depicted in Fig. 8, are *Choose Single action* and *Picks up the appropriate algorithms*. ALGOC implemented them in a generalized fashion. There is a WorkSpace Evaluator artifact that takes any specific criteria and pushes an answer into a pile of potential answers.
Perhaps one of the most significant variations in ALGOC is the concept of similarity managing. Percept properties are those aspects of percepts that need regular updating. In FLEXO, all percepts in a workspace should have the same properties, to provide a means for comparing them. In ALGOC, comparators must be defined in every case to overload the default comparison criteria. This way every specific model should have its own metrics to evaluate relevance and distance between percepts and concepts. The Workspace evaluator is the kernel of these metrics, but it could also assist Codelets if they need to evaluate a distance.
Generally speaking, the rest of the Codelet and percept treatment is very similar to the implementations described in the LIDA and FLEXO papers. The Workspace, Workplanes, and coderack are also very similar. The specific differences summarized here are the result of re-engineering the overarching idea to transform it into a framework.

5. FIC Implementation

This section presents the functionalities of FIC and general technical information. It describes the prototype, from technical overview to main functionalities to implementation.

5.1 Technical overview

The implementation of FIC currently requires Java, a minimum of of 50 Mb hard disk storage and, at least a 1.2 GHz processor. The overall architecture of the FIC prototype can be seen in Fig. 11. Software was defined with several Object Oriented Design Patterns to provide the source code with flexibility and readability. The robot is a LEGO Mindstorm NXT, with LEJOS 6.0., connected to the Real Time Controller (RTC) which interacts with FIC. As FIC was designed to be a framework, the actual RTC communicates to FIC the same way other devices do.
It is the responsibility of the RTC to provide correct initial information to and handle the World Map (WM). FIC will build an internal copy of this map within its local Dynamic Knowledge Database (DKD), and perform proper updates to it using data input the RTC. ALGOC is the core of the system. It is consistent with both FLEXO and LIDA functionality. What FIC does is perform basic data processing and submit translated information to the kernel. Then it activates ALGOC functionalities to let it translate concepts (i.e., receiving real world information and gathering knowledge about obstacles) and perform complementary information managing. FIC acts as a counterpart of the RTC that controls the robot. It works using a static map of the world (previously made and loaded as initial knowledge) and making decisions about robot activity. Eventually it will accept additional command information coming back from FIC. While the main storage for the RTC is the WM, the FIC's is the local DKD, which holds key information on life cycle management and specific data to improve performance. Unlike implementations such as NaviGates [27], the global architecture does split the controller into levels of processing. This way, RTC can perform its tasks independently with or without FIC.

5.1.1 Functional description

To achieve FIC functioning, the first step is to activate the initial previous knowledge about the problem domain. Then the RTC will start and notify FIC of its presence. When the robot starts,

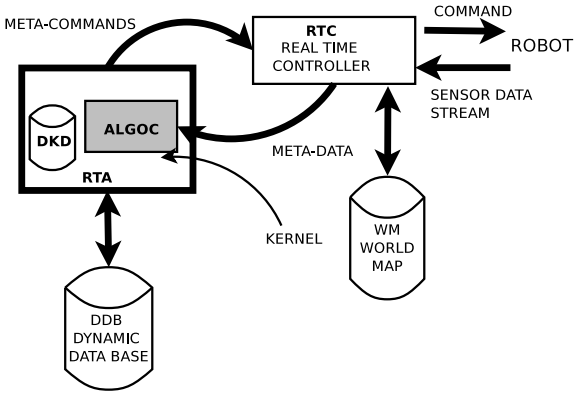

Fig. 11: FIC global architecture.

sensor streams are sent to the RTC, and are processed and used to update the initial map of the world (WM). Part of the information will be sent also to FIC to set up the initial status of the system, including updating the local Dynamic Knowledge Database (DBD). When the robot sends a data-stream to RTC, it answers with after a short time with a command to the robot, and sends data to FIC. When FIC detects a certain obstacle, it determines the type of concept corresponding to the obstacle. If there is no actual concept for the obstacle, it inserts a new concept and returns a meta-command denoting the obstacle as an unknown object, and assigning it a previously defined action (e.g., elude it). Along with this, FIC returns a number indicating the relevance of the RTA suggestion. In the case of a known concept, FIC uses available information and related concepts to select the best meta-command to be sent. RTC eventually receives the meta-command. Using the meta-command and the refreshed environment information, the RTC will decide whether or not to change the robots behavior. Why wasn't the RTC implemented as part of the LIDA model? It could have expected FIC to make control decisions. The answer is twofold:

- *Timing requirements:* As a real time activity, movement decisions must be made very quickly, which cannot be guaranteed with FIC. Concept searching and its internal processing would not give an answer if a concept learning or ambiguous situation arises.

- *Experience re-usage:* In the described approach there is a dual-loop feedback between the RTC and FIC. The first one, a close-loop, is a simple and traditional robot controller, while the second one acts as an upgraded version of the first, adding extra functionalities on demand from the first one.

5.1.2 General adaptations of conscious models

To implement robot behavior using the ALGOC model, it is convenient to follow up the Concepts Design Strategy. The main functionalities are:

1. Acquire data from the environment. Percepts are specialized concepts that represent external information. They are activated and adapted dynamically by several specific Codelets including Codelets illustrated in Fig. 8, among others.

2. Evaluate data (turn it into information). When percepts are processed by certain Codelets (for instance those in Fig. 9), new concepts are activated and fire other Codelets. This chaining emulates evaluation activities.

3. Perform a task. This is a composed task, because it consists of certain concepts in succession (due to its high activation level. Effectors are also Codelets that bring up candidate suggestions as answers. Afterwards, the RTC evaluates the resulting advice and its weighting and decides what command to do.

4. Move to a target place. The only defined concepts are go-ahead, turn-right, stop, distance, step-distance (meaning displacement at a single movement) and spin. All of them are activated according to the current status of the robot in the world map and the activated set of percepts.

5. Avoid certain dangers (e.g., collisions, self destruction, etc.). To prevent certain situations from occurring, obstacle type concepts have been defined but by themselves, they are not enough to model critical situations, which are handled by Codelets that try to recognize a predefined combination of circumstances. If they succeed, they will change the weighting of the advice given to the RTC, and change the activation logic.

5.1.3 Specific adaptations for mobile robots

As timing and certain types of information are specific to the field of application, there is a set of requirements that bias the implementation:

1. Fast recognition of known objects is mandatory. This requires that concepts related to obstacles get raised to a high activation value quickly, so Codelets can implement a different activation approach for them.

2. Obstacles have two predefined categories: targetable and forgettable. This sets up a predefined hierarchy problem in the definition of concepts.

3. Effectors are just recommendations (i.e., meta-commands) and do not act on the real world directly, only through the RTC. As a consequence, all effectors must provide the RTC with extra information for it to make its next decision. For the same reason, output recommendations come with a score. If the current RTC status requires a quick reaction and FIC outputs have low ranking, then the controller discards any suggestion. But if they have high priority, they will likely compete with the RTC commands, depending on the current activity and status of the robot.

4. The simplest implementation is the most desirable. It was a design decision to start with a cognitive map of the world. It contains the initial spatial cognition, or the robot environment. This way, initial concepts can be defined and activated to model obstacles and structure-building with a low degree of activation.

5. There are a set of known obstacles at initiation. They are described as congregations of other concepts representing information that describes them physically (e.g., number of non-free cells, relationships between occupied cells, etc.). They are handled by specific object-detector Codelets that detect certain aggregation of concepts that define an obstacle and object-classification Codelets that apply distance metrics to evaluate similarity to previously known objects.

6. There is a class of obstacle-similarity Codelets that implement specific metrics to asses each percept and its related active concepts and activate a concept that enables a newly encountered obstacle to be treated in the same way as a previously encountered obstacle.

7. The model should be able to learn about new obstacles. Thus, the ability to recognize unknown objects is required. They are detected by the distance metrics applied by the obstacle-similarity Codelets. Whenever a distance is evaluated to be beyond a parametric threshold, a learning-obstacle concept is activated. This is a special type of effector that asks for new structural information (Codelets, links, concepts) to learn how to manipulate new obstacles.

8. Recognize dangerous environments. Specific concepts and Codelets are designed to detect certain obstacles and map cells combinations. They are not currently implemented.

5.1.4 Current FIC Implementation

This section special focuses on the current FIC workflow. Fig. 12 shows an overview of the main subsystems defined for the robot localization system. The diagram shows main components of ALGOC with dashed lines. Solid lines indicate which components belong to the derived FIC robot

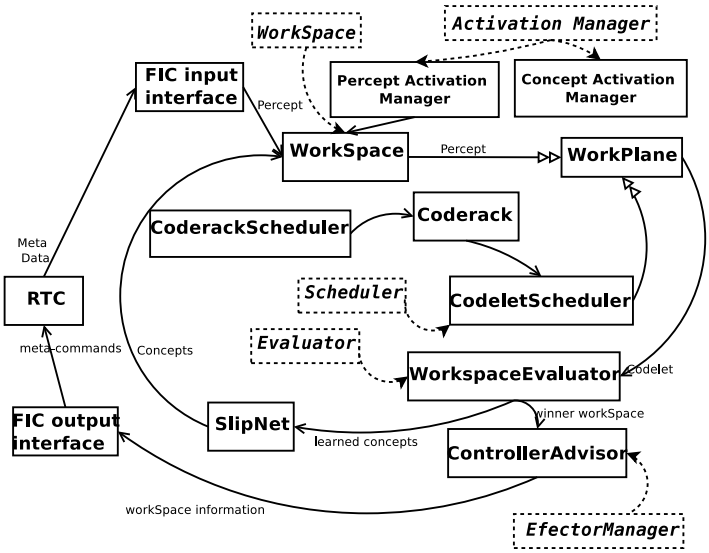

Fig. 12: FIC work-flow.

model. FIC implements a set of specialized functionalities that are useful for an autonomous-mobile robot and are not part of the ALGOC framework. Percepts are created from the meta-data sent by the RTC. The workspace is first populated with percepts, which are the unique conceptual information directly handled by it, because the model employs a hot-thread approach for percepts. The cycle is very simple. Starting in the RTC, data goes through the FIC input interface and enters the Workspace (WS). Every new Work-plane (WP) has percepts in it. The CodeletScheduler will manage the general activation approach for all Codelets, which will remain on a WP throughout. The WorkspaceEvaluator will select the best WP and push it into a pile. The ControllerAdvisor pops the best WP and extracts the proper information from it to feed to the FIC output interface. The ALGOC Evaluator is a general purpose comparator that was defined as the WorkspaceEvaluator. To perform any comparison, a subset of attributes is defined as relevant for both percepts and for every percept. It uses Euclidean distance to compare between numerical percepts and non-Euclidean Manhattan distance between the similar relevant attributes in concepts. When a concept does not have a relevant attribute, there is a default qualification.

There is also a similarity threshold, H, to determine whether a new but similar concept should also be treated as unknown. In these cases, a Codelet creates a special concept to alert the EAM (see Fig. 10) that new external knowledge is required (which will eventually require an operator interaction). As mentioned in section 4.4, this is one of the main artifacts of that ALGOC meta-model framework that is not defined in LIDA, IDA, or FLEXO. Note that even when the concept is not totally recognized, the model will keep working, replacing it with the closest current concept.

EAM is a general purpose module that also provides tools for issues specific to the current model (i.e., FIC), such as sampling parameters, mapping tasks to execution threads, accelerating Codelets according to the current devices being used, type of world-map, etc.

As noted in section 5.1), there are also two data storage sites: the local Dynamic Knowledge Database (DKD), that has all the knowledge that FIC requires at start-up (initial sampling speed, world-map, etc.) and the Dynamic Data Base (DDB), which contains many other long-term data for FIC (mainly related to pending tasks for the EAM module). Information is uploaded from the former during initialization of the Codelets, WS and WP. The later is used for off-line activities.

5.2 Performance tests

To evaluate the behavior of FIC under the ALGOC model, the close-loop controller (RTC) was tested first. Afterwards, the same robot was used with FIC advice. For simplicity, the obstacle, robot effectors and world-map were the same.

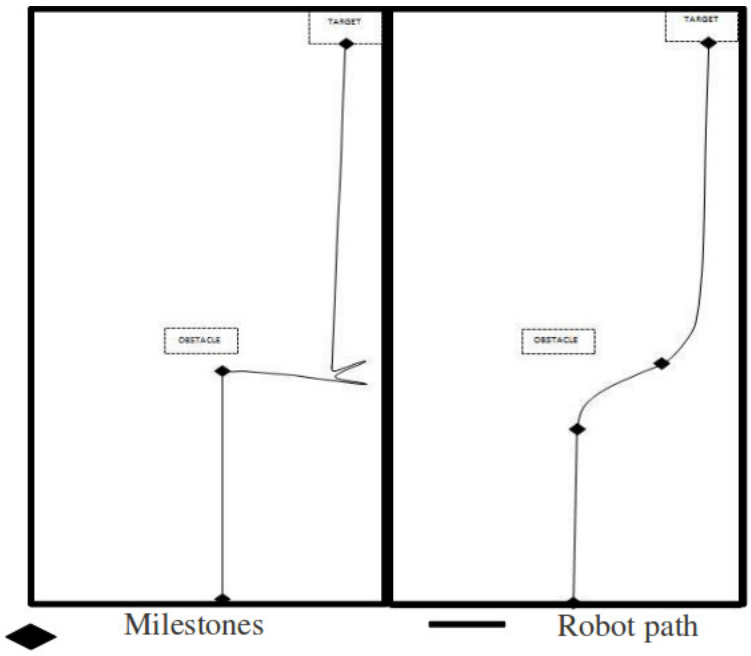

Fig. 13: Path with a single controller (left) and with FIC (right).

Because the close-loop controller in both tests is the same, the response time of the controller was not tested. FIC will answer when needed and not on the fly, because it is intended to advise only in certain complex situations and does not make every decision.

5.2.1 Testing with a single controller

The left side of Fig. 13 traces the path taken by the robot when the controller described in Fig. 1 is used. The test was performed in a 285 cm x 145 cm. rectangular room. It took 20.543 seconds from start for the robot to reach the goal. There was only one 12 cm x 30 cm rectangular obstacle situated in the middle of the room and the floor had no slope or steps.
The sampling rate for every input device in the robot was 40 kHz, and the wheel speed had a maximum speed of 21.05 cm/sec.
The robot took the path shown in Fig. 13, broken into 6 segments, with a total of 1.16 seconds in delays. The total path length was 383.6cm.

5.2.2 Testing using FIC

This second test was performed with the same robot hardware, an identical starting point, target and world-map.
The path is on the right part of Fig. 13. This time, it took the robot 16.382 seconds to reach its goal. The path was 296cm and had no milestones.
Although it is not the goal of this paper to evaluate the implementation, below are some statistics on the size and workload of FIC:

-Number of Codelets defined: 37
-Average activation or degree of learning reinforcement ([0..10]): 4.2
-Maximum number of workplanes: 10
-Average Number of workplanes: 4.7

6. Future Work

Traditional localization in places such as factory buildings and halls, usually involves processing a set of events and facts, such as the location of natural landmarks, walls, and dynamic obstacles. There are several approaches for solving localization problems [8] [1] [9]. The FIC architecture was developed to accomplish concept handling and meta-command interchange through dual-loop feedback. Because the computational requirements for FLEXO and LIDA are relatively high, the Controller and FIC hardware restrictions have been designed to suit a robot. Each one has an extra load of computation due to the dual-loop. The main coding work is complete. Debugging is mostly complete, although new, permanent code may be required to implement Codelets and concepts to best model specific situations. Application tuning is currently underway to optimize robot performance. ALGOC, which is inspired in the FLEXO and LIDA systems, is the core of the prototype, and has been adapted and optimized using a specific step by step design strategy. In comparison to FLEXO, there are new modules that implement part of LIDA to complement and extend FIC's abilities, but these are still undergoing tuning. Further improvements to the model will require inclusion of additional knowledge about the standard abilities in the robot (e.g., its ability avoid obstacles and danger in advance) and the surroundings (i.e., types of obstacles and the properties of such obstacles). It also would be interesting to include targeted obstacles as a subclass of the general obstacle concept. Alternatives activities such as paint, lift, run, and jump would complement the existing activities (advance, rotate and stop). In time, interesting additional extensions may be added, including time considerations related to obsolescence and past/future consciousness, improved object/obstacle recognition, and dynamic world-map building.

Acknowledgments

We would like to thank Dr. Joaquin Vanschoren for his help during the debug of FLEXO and Uma Ramamurthy for her help editing and reading the manuscript. We also like to thank David Trejo, Nelson Biedma and Lucas Rancez for their advice and help in the implementation of FIC. Finally, we'd like to recognize the support of Universidad de Palermo during our project.

References

[1] D. Lecking, O.Wulf, and B. Wagner, *Localization in a wide range of industrial environments using relative 3D ceiling features.* Emerging Technologies and Factory Automation, IEEE International Conference on ETFA 2008. pp 333 - 337, 2008.

[2] B. J. Baars, *A Cognitive Theory of Consciousness.* Cambridge: Cambridge University Press, 1988.

[3] B. J. Baars, and S. Franklin, *How conscious experience and working memory interact.* Trends in Cognitive Science 7:166172, 2003.

[4] A. D. Baddeley, and G. Hitch, *The psychology of learning and motivation: Advances in research and theory.* Working memory. In G.H. Bower (Ed.) (Vol. 8, pp. 47–89). New York: Academic Press., 1974.

[5] S. Franklin, and F. G. Patterson, *The LIDA architecture: Adding new modes of learning to an intelligent, autonomous, software agent.* IDPT-2006 Proceedings (Integrated Design and Process Technology)., 2006.

[6] D. R. Hofstadter, and M.Mitchell, *The Copycat Project: A model of mental fluidity and analogy-making.* In Holyoak, K.J. & Barnden, J.A. (Eds.) Advances in connectionist and neural computation theory, Vol. 2: Analogical connections. Norwood, N.J.: Ablex., 1994.

[7] H. F. Ebel, C. Bliefert, and W. E. Russey, *The Art of Scientific Writing : From Student Reports to Professional Publications in Chemistry and Related Fields.* Weinheim: Wiley-VCH Verlag GmbH & Co. KGaA, 2004.

[8] S. Kolski, D. ferguson, M. Bellino, and R. Siegwart, *Autonomous Driving in Structured and Unstructured Environments.* IEEE Intelligent Vehicles Symposium, 2006.

[9] C. Eberst, M.Andersson, and H.I. Christensen, *Vision-based door-traversal for autonomous mobile robots.* Inst. for Real-Time Comput.-Syst., Tech. Univ. Munchen; Proc. 2000 IEEE/ RSJ International Conference on Intelligent Robots and Systems., 2000.

[10] H. Zendera, O. Martnez Mozosb, P. Jensfeltc, G.Kruijffa, and W. Burgardb, *Conceptual spatial representations for indoor mobile robots.* German Research Center for Artificial Intelligence (DFKI GmbH). Elsevier., 2008.
[11] D. L. MacPherson, *Automated Cartography by an Autonomous Mobile Robot Using Ultrasonic Range Finders.* GPhD Thesis.Naval Postgraduate School, Monterey CA., 1993.
[12] S. Franklin, and A. Graesser, *Is it an Agent, or just a program?: A Taxonomy for Autonomous Agents.* Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages, Springer-Verlag., 1996.
[13] S. Franklin, *An Autonomous Software Agent for Navy Personnel Work: a Case Study.* In Human Interaction with Autonomous Systems in Complex Environments: Papers from 2003 AAAI Spring Symposium, ed. D. Kortenkamp, and M. Freed. Palo Alto: AAAI., 2003.
[14] V. Matelln, and J. Caas *WiFi localization methods for autonomous robots.* Robotica 24(4): 455-461, 2006.
[15] J. B. Hayet, and F. Lerasle, M. Devy *A Visual Landmark Framework for Indoor Mobile Robot Navigation.* IEEE International Conference on Robotics and Automation. Proceedings. ICRA, 3942-3947 vol.4, 2002.
[16] J. S. Gutmann, and D. Fox *An Experimental Comparison of Localization Methods Continued.* IEEE/RSJ International Conference on Intelligent Robots and Systems, 454-459 vol.1, 2002.
[17] L. McCauley, and S. Franklin, *A Large-Scale Multi-Agent System for Navy Personnel Distribution.* Connection Science 14:371-385., 2002.
[18] A. H. Varela, E. Thompson, and E.Rosch, *The Embodied Mind: Cognitive Science and Human Experience.* Cambridge, MA: MIT Press., 1991.
[19] L. W. Barsalou, *Perceptual symbol systems.* Behavioral and Brain Sciences, 22, 577-609., 1999.
[20] A. M. Glenberg, *What memory is for.* Behavioral and Brain Sciences, 20, 1-19., 1997.
[21] K. A. Ericsson, and W. Kintsch, *Long-term working memory.* Psychological Review, 102, 211-245., 1995.
[22] J. L. Posadas, J. L. Poza, J. E. Sim, G. Benet, and F. Blanes, *Agent-based distributed architecture for mobile robot control.* Engineering Applications of Artificial Intelligence, 21(6), 805-823., 2008.
[23] M. A. Conway, *Sensory-perceptual episodic memory and its context: Autobiographical memory.* In A. Baddeley, M. Conway, & J. Aggleton (Eds.), Episodic Memory. Oxford: Oxford University Press., 2001.
[24] S. Franklin, B. J. Baars, U. Ramamurthy, and M. Ventura, *The Role of Consciousness in Memory.* Brains, Minds and Media1:1-38., 2005.
[25] B. J. Baars, S. Franklin, *Consciousness is computational: The LIDA model of global workspace theory.* International Journal of Machine Consciousness, 1, 1: 23-32., 2009.
[26] H. P. Nii, *The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures.* The AI Magazine, Summer 1986, 3853., 1986.
[27] R. Knotts, I. R. Nourbakhsh, and R. C. Morris, *NaviGates: A Benchmark for Indoor Navigation.* ACE publications. pp. 36-42., 2007.
[28] S. A. Sloman, *Cognitive Architecture.* InWilson, R., & Keil, F. (Eds.), The MIT Encyclopedia of Cognitive Science, pp. 124-126. MIT Press, Cambridge, MA., 1999.