14

A Decentralised Software Process Approach For Real time Navigation of Service Robots

S. Veera Ragavan and Velappa Ganapathy
Monash University, Sunway Campus, Malaysia

1. Introduction

Convergence of Communication, Computing and Control is considered by many as the future of Information Technology. In the same breadth, the recent advances in Key Technology such as Telematics, M2M, Smart Wireless Communication Devices (SWCD) and Sensor Networks (WSN) from Telecommunications' domain; Artificial Intelligence (AI), Sensor Fusion , Behavior Fusion and Hybrid algorithms (ANFIS, GA) from the Robotics Domain; Open Source Systems (OSS), Geographical Information Systems (GIS), Service Oriented Architecture (SOA) and Session Initiated Protocol (SIP) from the Information Technology Domains necessitates a paradigm shift in approaches to building applications for future Robots.

Early single function robotic systems were designed as control systems with a clear feedback model. A Sensor generates feedback, and any deviation from the expected feedback generated by the Sensors updates a control signal to minimize the error over time. As complexity of modern day Service Robots grew and the robots needed to perform multiple functions, the perception-action loop was extended linearly to include a planning component resulting in the sense-plan-act paradigm. Obviously it does not perform very well in dynamic and unpredictable environments: the sensors and real world models are usually inadequate as the actions are not always a direct consequence of perception.

The hybrid deliberate/reactive approach has proven very successful, practical and robust in a large number of implementations, and it appears that there is general agreement among the research community that this is the best type of architecture for Service Robots. However, some types of modules are hard to force into any particular layer. In a general framework, it is imperative that no special architecture is preferred for enforcement and a good support for builders of the hybrid deliberate/reactive architecture is important so that the framework supports parallel execution of behaviours. This is precisely where the proposed architecture scores above the other architectures.

The major problems for robotics today lie, not in the hardware but on the software side. There are plenty of well functioning and robust algorithms developed by competent researchers readily available. Each new implementation would provide significant gains in the performance and capabilities, but it will be lost due to non portability and reuse issues.

While the lowest layer or reactive layer has to be embedded on the robot controller due to the obvious fact that this layer requires the highest response and lowest CPU time, in our approach the Middleware layer helps us to switch based on deliberative approaches from the repository of allowable robot behaviours. What was essentially an traditionally AI Rule Based Behaviour Switching now graduates to a Location Based Behaviour Switching based on a host of Soft computing techniques .

Most of the commercial Service Robotics industries even today, employ at best the "earlier" or traditional approaches to software reuse, which are built on the paradigm of a set of libraries containing many small building blocks. Some of the more advanced establishments use object-oriented frameworks resulting in a generic design that can be instantiated for each object system constructed. While the Object oriented framework ideally captures the elements common to a family of related systems, it is essentially a large design pattern capturing bulk of the system functionality in one specific kind of object system, which is maintained as a single entity. Each software system using framework is only an instantiation of that framework. Traditionally, control of generalised Service Robots required well-defined activities tightly coupled across different hierarchies on single platform i.e. monolithic control architecture.

In a distributed system or multiprocessor, Middleware allocates system resources, giving requests to the operating systems on the individual processors to implement those decisions. One of the key differences between Middleware and Software Libraries are that Middleware manages resources dynamically. In a uniprocessor, the operating system manages the resources on the processor (for example, the CPU itself, the devices, etc.) and Software Libraries perform computational tasks based on those allocations.

By Process we mean a system element that is independent, and can be freely deployed and versioned. This approach loosely couples the various layers into process components that are well defined entities that can be replaced or made redundant without affecting the rest of the systems. It is shown here how they can be developed and tested separately and integrated later building on the Middleware Framework to provide a systematic approach to developing software that would be easy to integrate, manage, and reuse

The field of robotics relies heavily on various technologies such as Mechatronics, computing systems, and wireless communication. Given the fast growing technological progress in these fields, robots can cater to a wide range of applications. However real world integration and application development for such a distributed system composed of many robotic modules and networked robotic devices is very difficult. Therefore, Middleware services provide a novel approach offering many possibilities and drastically enhancing the application development for robots.(Mohamed, Al-Jaroodi et al. 2008)

Middleware is prevalent in IT-oriented server systems than in Embedded Systems. Several Embedded Middleware Systems have been developed based on IT standards but with heavy footprints. Middleware architectures are slowly and steadily evolving to support embedded operations. A Telematics device Middleware and the GPRS link enable configuring and reconfiguring the devices as many times as required and this is effectively used to change the control program and actions remotely to use the robot for different services without having to make any major changes to the design, hardware or software.

This paper discusses our domain, existing architectures, component and processes execution techniques and the approach we took to integrate these to form a distributed decentralised web enabled service that is robust and safe-fail. The resulting architecture is simple, and can

support a wide range of trade-offs that can be manipulated easily at run-time based on control of processes rather than control of discrete actions. The current approach is a loosely coupled integration of different process technologies and computational mechanisms.

2. The Service Robot Domain

The Services area is just emerging as the future of field application and Services computing has been recognized as a new foundational discipline of the modern services industry(Zhang 2008). Even though the number of Service Robots currently in operation is small, the number of people working in the service field shows a constant growth rate. Consequently, there is an enormous potential for the expansion of this sector. Due to often strict requirements with respect to safety (personal and functional) associated with robot autonomy and navigation in partially or entirely unknown environments (coupled with the convergence issues) the use of robots for carrying out service tasks has been very limited. Only a fraction of existing services which incorporate handling, transportation and working can be automated. A successful design of future Service Robots will be based upon detailed knowledge of available technologies and methodologies to design handling devices or mobile platforms, peripheral devices, and organizational schemes which account for a flexible, fault-tolerant and user friendly human-machine interaction. (Schraft 1994)

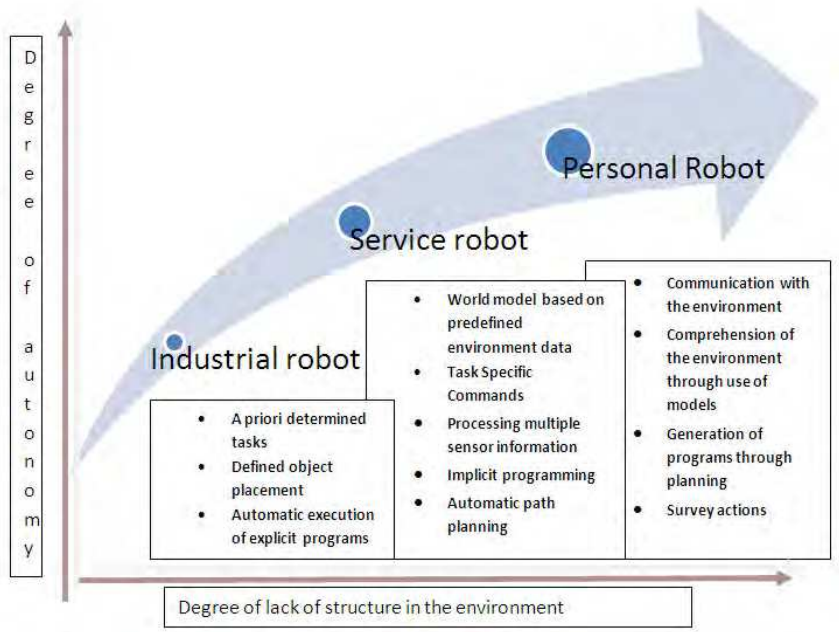

Fig. 1. Characteristics of Industrial Service and Personal Robots [based on (Schraft 1994)]

Such a Design philosophy calls for architecture that must also provide a structure that improves the coherence and modularity of the system, while supporting large scale robotic system development which include considerations of real-time response and a layered

decomposition of the system to distinguish the granularity of the modules responsibilities. Service Robots must have highly competent reactive mechanisms to be safe, flexible and easy to use. At the same time, planning and sequencing are useful to reduce the repetition and taxation on the user for direction. (Kawamura, Pack et al. 1995).
The uncertainties from the environment, the complexities of software/hardware interactions, and the variability of the robotic hardware make the task of developing robotic software complex, hard, and costly. Hence, it has become increasingly important to leverage robotic developments across projects and platforms.(Nesnas, Wright et al. 2003)
In spite of an explosion of technology and methods, the Service Robots are still not complex and in their early stages of development. Many researchers specialize in one or more areas/topics, which usually involve development of algorithms. However, in order to test the competence on a real robot, a complete system is needed involving a process based approach. Many of these are required to run in parallel and need to communicate both synchronously and asynchronously. It has to also accommodate changing application requirements, incorporate new technology, interoperate in heterogeneous environments, and maintain viability in changing environments. This puts a tremendous burden on the developer if he or she has to build everything from scratch and hence a delay in "Market ready" products. Hence, it has become increasingly important to develop Service Robots on General Platforms and Frameworks. (Ragavan and Ganapathy 2007).
We present a Novel Decentralised Architecture for Navigation and Control of Service Robots based on control of processes rather than control of discrete actions. The current approach is a loosely coupled integration of different process technologies and computational mechanisms. It is our firm contention that a well designed software architectural framework is necessary to effectively leverage microcontrollers (Read Service Robots), wireless networks (read Telematics, distributed wireless networks) and process orchestration (read service) to address problems of complexity, scale and reliability of networked Service Robots

a. Layered Architecture and Hybrid approaches

Early robotic systems for single functions were designed as control systems with a clear feedback model. A Sensor generates feedback, which is compared to the expected feedback derived from a model of the system. Any deviation is used to update the control signal so as to minimize the error over time. As complexity grew and the robots needed to perform more than one function, the perception-action loop was extended to have a planning component. This was a natural linear extension beyond traditional control towards modern day Service Robots. This resulted in a hierarchical system having an elaborate model of the world, using sensors to update this model, and to draw conclusions based on the updated model. Obviously it does not perform very well in dynamic and unpredictable environments as the sensors and real world models are usually inadequate. That the actions are not a direct consequence of perception is perhaps the reason why it is also called the sense-plan-act paradigm.
Reactive approaches are often capable of autonomously exploring new regions in the environment and, as there is no fixed plan, they are generally able to respond rapidly to any changes that may occur in the operating environment. Moreover, they are more tolerant to uncertainties in sensor measurements and the errors. Robots that were running reactive behaviour based systems performed very well, also in changing environments. However, the purely reactive scheme is not capable of performing complex tasks. A software

architecture based on purely reactive approach is usually monolithic and requires rewriting of control software for even small changes in the task, or environment.
On the other hand deliberative navigation methods generally assume that the obstacles in the environment in which a robot moves are known in terms of their physical location and dimensions. The navigation task is then to plan a path that is both collision free and satisfies certain optimization criteria. The classical deliberative approach to navigation is based entirely on planning and on explicit symbolic models of the world exhausts the computation resources all along the way (Brooks 1991). Even more, it does not seem to operate successfully in a dynamic changing world. It has difficulties in dealing with sensors' errors as well. The models it uses are not realistic; it appears that the world is too complicated to be presented completely. Attempts to create a complete model that includes all the essential knowledge needed to deal with the uncertainties and surprises of the real world, became enormously big and the planning too expensive in time and computer resources. Hence, it has become increasingly important to leverage upon Hybrid Approaches to robotic developments across projects and platforms.

b. Hybrid Approaches

A hybrid approach, combining low-level reactive behaviours with higher level deliberation and reasoning, has since then been common among researchers (Arkin 1990; Cattoni, Di Caro et al. 1994). The hybrid systems are usually modelled as having three layers as shown in Figure 1; one deliberative, one reactive and one middle layer (Gat 1992) and this approach for a long time now remains vastly unchallenged.
There is also a sound architectural rationale for having exactly three major components and not just because three is an aesthetically pleasing number. It has to do with the role of internal state and with ability to organize algorithms according to whether they contain no state, contain state reflecting memories about the past, or contain state reflecting predictions about the future. Abstractions are then used to isolate aspects of reality that can be tracked or predicted reliably, and to ignore other aspects (Erann 1998).

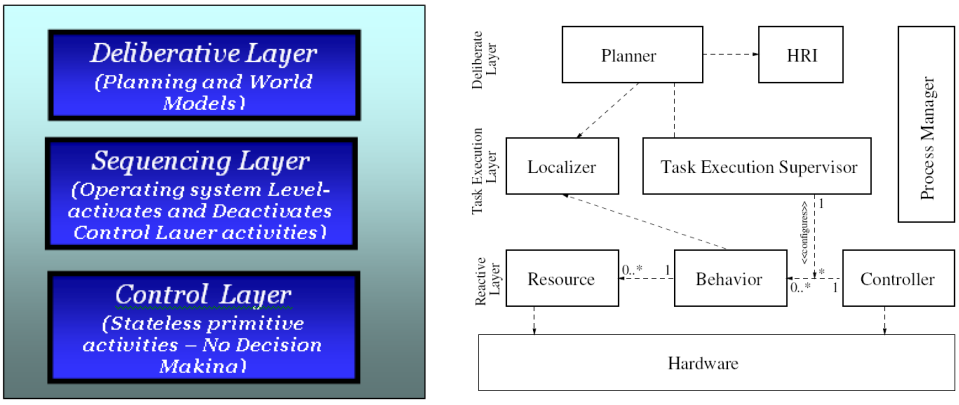

Fig. 2. Three Layer architectures - ATLANTIS (Gat 1992) and BERRA (Lindstrom, Oreback et al. 2000)

The Lowest Layer or control layer is mainly reactive with no decision making and the process computations at this layer use the least amount of CPU time and are tightly coupled to the Sensor-Actuator layer. The Middle level or grey layer bridges the gap between the two layers (Cattoni, Di Caro et al. 1994) and is usually either a Rule Based Layer or a Finite State Machine deciding which set of behaviours should be running. It also acts as a supervisory layer and catches failures of the reactive layer and then executes deliberative plans. The highest level of activities like planning and world modelling are done at this level and these are the areas that need significant computing resources. The advances in distributed computing techniques and communication infrastructure are leveraged in the proposed architecture to offer a decentralized control system.

3. Decentralised Hybrid Software approach

The hybrid deliberate/reactive approach has proven very successful, practical and robust in a large number of implementations, and it appears that there is a general agreement among the research community that this is the best type of architecture for Service Robots. However, some types of modules are hard to force into any particular layer. In a general framework, it is imperative that no special architecture is preferred for enforcement and a good support for builders of the hybrid deliberate/reactive architecture is important so that the framework supports parallel execution of behaviours. This is precisely where this proposed architecture scores above the other architectures.

The major problems for robotics today lie, not in the hardware but on the software side. There are plenty of well functioning and robust algorithms developed by competent researchers readily available(Ibrahim and Fernandes 2004). Each new implementation would provide significant gains in the performance and capabilities but it will be lost due to non portability and reuse issues.

While the lowest layer or reactive layer has to be embedded on the robot controller due to the obvious fact that this layer requires the highest response and lowest CPU time, the Middleware layer helps us to switch from the repository of allowable robot behaviours. What was essentially an AI Rule Based Behaviour Switching now graduates to a Location Based Behaviour Switching in the current architecture.

In contrast to the "earlier" or traditional approaches to software reuse, which are built on the paradigm of a set of libraries containing many small building blocks, object-oriented frameworks allow the highest common abstraction level between a number of similar systems to be captured in terms of general concepts and structures. The result is a generic design that can be instantiated for each object system constructed(Lewandowski 1998). The Object oriented framework(Bagchi and Kawamura 1992) , (Miller and Lennox 1990) (Chochon 1993) is ideally suited for capturing the elements common to a family of related systems. In this sense, the framework is essentially a large design pattern capturing the essence of one specific kind of object system. The bulk of the system functionality is captured in the framework, which is maintained as a single entity. Each software system using framework is an instantiation of that framework (Lewandowski 1998). Object and component frameworks can also be seen as a special breed of object-oriented systems — they are extensible, semi-finished pieces of software. Completing the semi-finished software leads to different software pieces, typically specific applications, that share the same core.

Though frameworks have been developed for a wide range of domains, they use common construction principles(Marcus, Wolfgang et al. 2000)

4. Overview of Software approaches in Robotics

Overview of some relevant software systems (implemented architectures) can be found in (Mohamed, Al-Jaroodi et al. 2008) (Utz, Sablatnog et al. 2002) . Examples of existing systems are AuRA(Arkin 1990; Arkin and Balch 1997), Task Control Architecture(Simmons 1994), Saphira, Teambots, Smartsoft, Mobility, Player/stage , MIRO(Utz, Sablatnog et al. 2002), LICA (Hu, Brady et al. 1995), ORCA (Oreback and Christensen 2003), BERRA (Lindstrom, Oreback et al. 2000), PeLote (Ruangpayoongsak, Roth et al. 2005) and Loosely coupled Layered architecture for Robot Autonomy CLARAty.(Volpe, Nesnas et al. 2001; Urmson, Simmons et al. 2003)

5. Middleware approach

In a distributed system or multiprocessor, Middleware allocates system resources, giving requests to the operating systems on the individual processors to implement those decisions. One of the key differences between Middleware and Software Libraries are that Middleware manages resources dynamically. In a uniprocessor, the operating system manages the resources on the processor (for example, the CPU itself, the devices, etc.) and Software Libraries perform computational tasks based on those allocations. Real world integration and application development for such a distributed system composed of many robotic modules and networked robotic devices becomes very difficult without Middleware. Middleware is prevalent in IT-oriented server systems than in Embedded Systems. Several embedded Middleware systems have been developed based on IT standards but with heavy footprints. Middleware architectures are slowly and steadily evolving to support embedded operations (Schmidt 2002). Middleware is software infrastructure that has been used to successfully integrate and manage software for complex distributed systems (Baliga, Graham et al. 2004; Baliga, Graham et al. 2004). Middleware is generally constructed to provide communication between application software and processes in P2P, client-Server or Publish - Subscribe models. Most Middleware address a particular domain such as Web Services, RTOS etc and define simple and uniform architectures for developing applications in that domain. Standard mechanisms for defining software interfaces and functionalities encourage the development of well-defined and reusable software. The Middleware concepts introduced make implementing the control systems more flexible so that they can be dynamically reconfigured with ease and can be upgraded or adapted in a flexible manner. An appropriate Middleware would allow software components to be integrated easily and provide standard functionalities such as support for robustness and fault tolerance, which can be easily reused in most applications.

Therefore, Middleware Services provide a novel approach offering many possibilities and drastically enhancing the application development for robots. We present a novel decentralized architecture as shown in Figure 2 for navigating and controlling a Service Robot based on control of processes rather than control of discrete actions.

By Process we mean a system element that is independent, and can be freely deployed and versioned. This approach loosely couples the various layers into process components that

are well defined entities that can be replaced or made redundant without affecting the rest of the systems. It is shown here how they can be developed and tested separately and integrated later building on the Middleware Framework to provide a systematic approach to developing software that would be easy to integrate, manage, and reuse.
At the core of the framework lies a Smart Wireless Communication device (SWCD) operating on Middleware. The SWCD Telematics framework based design accommodates three dimensions of variability namely:

- varying operating systems,
- varying hardware platforms, and
- varying implementation languages.

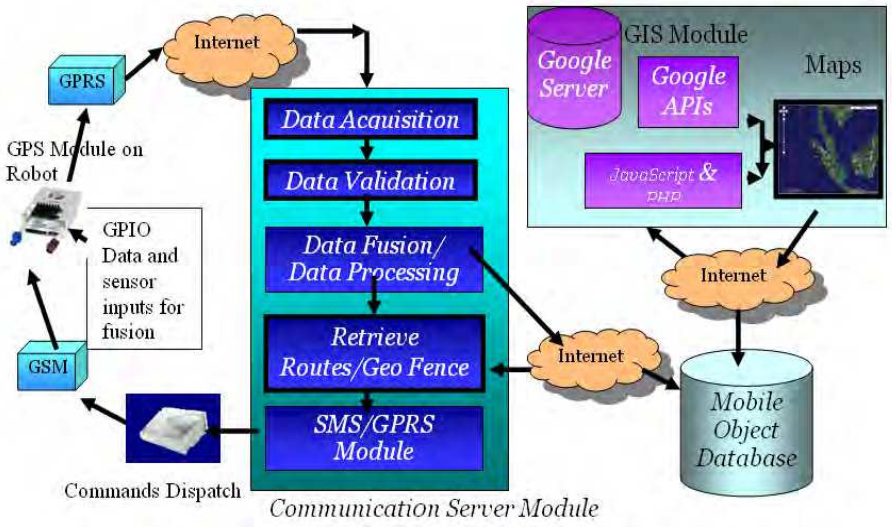

Fig. 3. SWCD based Decentralized System Architecture
Since the framework does a mapping from a constant high-level API to a set of native APIs of different operating systems and hardware platforms, only few interactions are sufficiently complex to warrant dynamic modeling. This static management of variables (as opposed to run-time dependencies) is sufficient since OS, platform, and language dependencies can all be resolved at build time. This still allows for later variability with respect to the features used in an application since the latter will only include those portions of the framework that are actually required and included in a given hardware configuration.(Marco 2003)

6. Implementation, Integration and Experimental Test setup

The implementation of the distributed software engineering concepts introduced in the earlier section permits our Service Robot application to be dynamically reconfigured with ease and to be upgraded or adapted in a flexible manner using the P2P, Client Server and Producer-Consumer models.

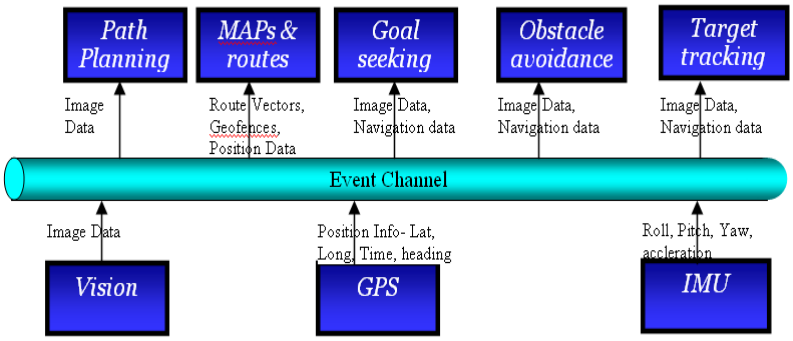

Fig. 4. Producer Consumer Middleware Model

The robot communicates to communication server (GPRS Data Acquisition Module, Data Validation and Command Dispatch Modules located in different physical locations through a secure web connection) in a P2P mode using GPRS over TCP/IP. We utilized Falcom StepIII Telematic modules (STEPPIII 2009) with Middleware (PFAL commands) to dynamically configure and process the sensor modules like GPS units, distance sensors and video camera. The communication between the Telematics terminal and the robot GPIO's (General Purpose Input and Outputs) is shown in the Figure 4-6.

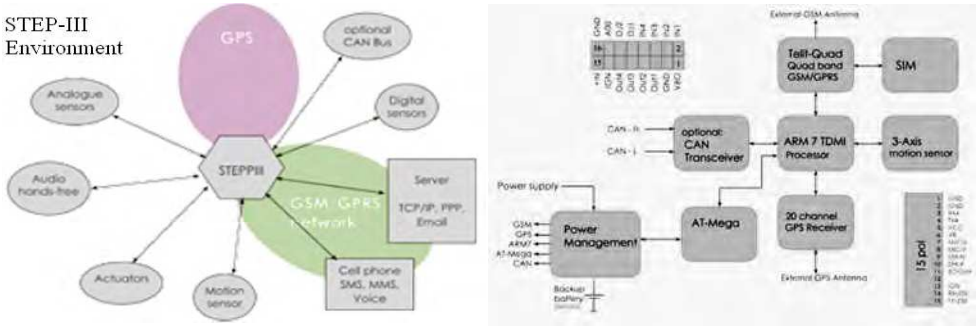

Fig. 5. Telematic unit – Function Blocks in Falcom - Step III.

The Robot communicates with the sensors through the event channels in the Publish – Subscribe mode through the GPIO's. The control components are software modules that perform the tasks of path planning, goal seeking, obstacle avoidance target tracking and localisation. The sensor components consist of device drivers and hardware. The sensors used here are GPS, vision systems and IMU and there exists a loose coupling through the Middleware providing an abstract communication channel referred to in the Figure 4 as Event Channel. The sensors register with the event channel as Publishers of data (e.g. camera as image data and GPS as position data) and Process components (e.g. Obstacle Avoidance and Target Tracking) are Subscribers. Subscribers get the data from whatever is available from the Publishers and new Publishers can be added at will.

7. Component Decomposition

The data flow diagram of the communication server is shown in the picture below. Many SWCD's can communicate to the communication server.

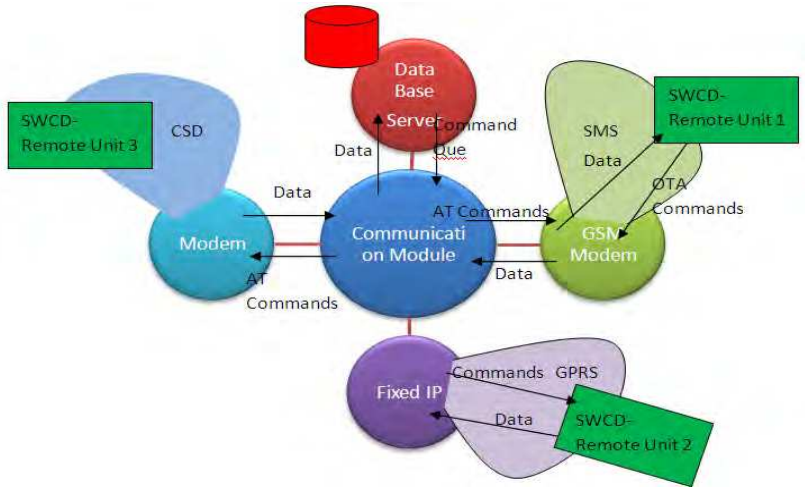

Fig. 6. Data Flow diagrams

The way-point and alert management processes are not shown separately in the picture as they form a part of the communications module. In addition to acting as a gateway for incoming and outgoing data communications, the communications module also acts as a monitor for alerts. Events monitored on the server side like way-points, SWCD not-reporting events, Geofence alerts, etc are handled by the communications module based on the live data feeds from the SWCD and hence, this component is an always-on module. As seen in the diagram, the communications module communicates with the SWCD's using either TCP over GPRS (preferred) or it falls back to SMS (over GSM modem). In case history data needs to be read from a remote unit in the absence of a GPRS network, the module does a data call to the SWCD via the modem.

8. GPRS Data Acquisition Module

A heterogeneous asynchronous communication process spread over four process layers and two physical layers is at the core of this design process as shown in Figure 7. The Communication Server Module was developed based on Client/Server architecture to acquire the GPS data over a GPRS network using a TCP/IP connection. In regions of poor GSM coverage the module switches to SMS for command transfer. GPS devices running on GSM/GPRS SIM cards were configured as clients to stream positional information to a Test Communication Server which had to be located external to the university network due to Static IP/Firewall restrictions. The units streamed data directly from GPS devices to an external communication server which performed the processes of data acquisition and validation functions before passing on the data for Data Fusion.

The remaining process of the Communication Services such as data and alert processing, command and alert service responses and configuration despatches was spread over a remote system within the university campus through the web. Therefore, data was collected externally, and the client application was used to stream the bulk data to the Server for processing.

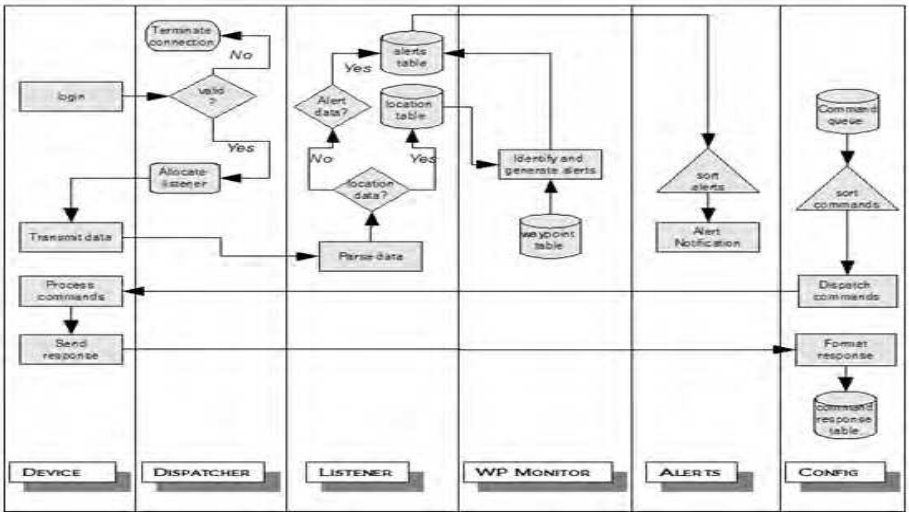

Fig. 7. Communication Process flow

Data received at the server was validated prior to structuring based on the starting characters of the string ($GPRMC) and by checking whether the third element in the string represents character 'A', which implies the validity of the string. Once validated, each string was structured using Sensor Bridge components.

ID	AsText(LATLON)	TIME	DATE	SPEED
1	POINT(3.07004 101.59712)	05:57:59	8:9:2005	31
2	POINT(3.06996 101.59712)	05:58:09	8:9:2005	0
3	POINT(3.06896 101.59708)	05:58:15	10:9:2005	20
4	POINT(3.06896 101.59708)	05:58:24	10:9:2005	48
5	POINT(3.06896 101.59708)	05:58:24	10:9:2005	48

Fig. 8. Spatial Table containing GPS data

Structuring Data and Data fusion is done using Sensor Bridge. Sensor Bridge is a component suite developed for Visual Studio 2005. It can incorporate data from different types of sensors and actuators. It consists of a hierarchy of class structures designed to manipulate raw sensor data received. Using Sensor Bridge, data received from different GPS devices were structured into separate series tables created from Sensor Bridge Components. Thereafter, the useful information such as latitude, longitude, time, date and speed were extracted to be stored for further processing. The inertial measurement readings obtained through the GPIO of the GPS modules are also streamed along with the position info to provide for near real time location awareness.

9. Mobile Object Database and Database Management

A Mobile Object Database (MOD) system was developed using MySQL as the data repository to store the processed GPS data. MySQL is an open source database management system, noted mostly for its speed, reliability and flexibility. Furthermore, MySQL incorporates spatial extensions under the specification of the Open GIS Consortium (OGC), which is an organization that groups many other organizations that prescribe standards for GIS data processing. The Mobile Object Database for this system was developed adhering to the structure of the geometry types proposed by the OGC. Figure 8 depicts the structure of a spatial table created for a GPS device using geometry type 'POINT' to store latitude and longitude values.

DATE	LAT	LON	SPEED
8:9:2006	3.084960	101.592280	89.000000
8:9:2006	3.084480	101.594440	90.000000
8:9:2006	3.083240	101.601320	85.000000
8:9:2006	3.083440	101.600040	81.000000
8:9:2006	3.083280	101.602120	81.000000
8:9:2006	3.084000	101.612320	35.000000

Fig. 9. Filtered and structured Position Information

A database connection between the MIS process and MySQL was established using a .NET connector component provided by MySQL. Data processed and structured into series tables using Sensor Bridge components were written into separate spatial tables to manage and store Geo-fences and Route patterns that have either been acquired through reactive navigation or through route patterns marked on the GIS system as shown in Figure 9, which can be re-transmitted to the Mobile robot through GPRS in order to navigate objects successively.

10. Transmission of Routes and Virtual Boundaries

Geo-Fences are virtual boundaries the robot is supposed to remain within to reach the goal and /or keep away to avoid dangers. Geo-Fences and the route vectors saved in the MOD are retrieved from the application to be transmitted back to the corresponding Mobile devices as shown in Figure 10.

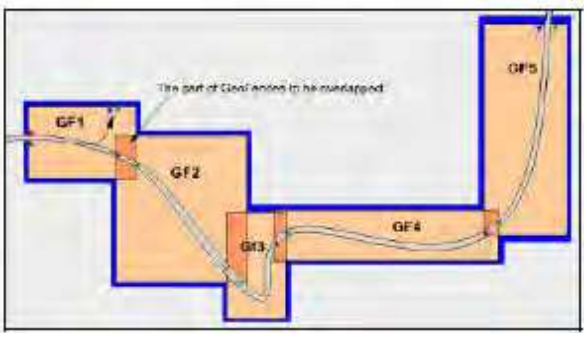

Fig. 10. Route planning and Geo-fence configuration "Over The air"

The Geo-Fences can also be created from the Maps like Google Earth and the boundary information can be sent to the Robot through the Telematic unit. PFAL (Device Middleware) commands were used to configure predefined routes and virtual boundaries in the GPS devices. For route configuration, a virtual boundary was created within a 30m radius for each waypoint in the route as required by the PFAL commands. Figure 10 depicts the configuration of a route to be transmitted to GPS devices

11. Over The Air (OTA) Configuration and GPIO Enabling and Disabling

The Device Middleware and the GPRS link enable configuring and reconfiguring the devices as many times as required and this is effectively used to change the control program and actions remotely to use the robot for different services without having to make any major changes to the design, hardware or software.

The feature diagram of a Smart Telematics Platform at various levels of abstraction as applied to a functional Product design is shown below. Middleware framework provides the maximum flexibility as the underlying SWCD hardware Platform is variable with respect to its assortment of components.

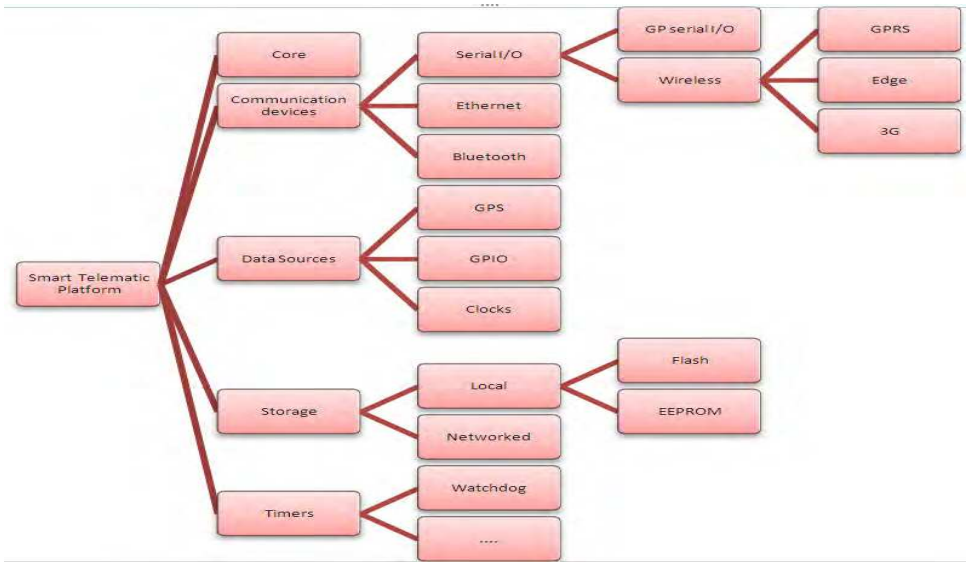

Fig. 11. Variable assortment of components form the variable SWCD hardware Platform

Sample configuration scripts sent over the air to configure, activate and connect to the device is shown below in Figure 12.

```
$PFAL,CNF.Set,PPP.AUTOPING=0,36000000
$PFAL,CNF.Set,PPP.PASSWORD=
$PFAL,CNF.Set,PPP.USERNAME=
$PFAL,CNF.Set,PROT.AREA=0
$PFAL,CNF.Set,PROT.BIN=0
$PFAL,CNF.Set,PROT.GGA=0
$PFAL,CNF.Set,PROT.GLL=0
$PFAL,CNF.Set,PROT.GSA=0
$PFAL,CNF.Set,PROT.GSM=10
$PFAL,CNF.Set,PROT.GSV=0
$PFAL,CNF.Set,PROT.IOP=0
$PFAL,CNF.Set,PROT.RMC=10
$PFAL,CNF.Set,PROT.START.BIN=$!!
$PFAL,CNF.Set,PROT.VTG=0
$PFAL,CNF.Set,TCP.CLIENT.ALTERNATIVE=0,217.1
$PFAL,CNF.Get,TCP.CLIENT.CONNECT=1,80.237.15
$PFAL,CNF.Set,TCP.CLIENT.DNS.TIMEOUT=86400
$PFAL,CNF.Set,TCP.CLIENT.LOGIN=1
$PFAL,CNF.Set,TCP.CLIENT.PING=1,300000
$PFAL,CNF.Set,TCP.CLIENT.SENDMODE=0
$PFAL,CNF.Set,TCP.CLIENT.TIMEOUT=300000,3000
$PFAL,CNF.Set,TCP.SMTP.CONNECT=0,<mailserver
$PFAL,CNF.Set,TCP.SMTP.FROM=<valid mailadres
$PFAL,CNF.Set,TCP.SMTP.LOGIN="<domain>",180,
```

Fig. 12. Middleware configuration scripts sent over GPRS and SMS

The GPIO's data requests can be enabled or disabled "on the fly" through Over The Air Commands to optimise on the performance of the mobile device that is resource and energy starved. Video camera for instance which consumes huge amounts of battery power can be switched ON/OFF or at optimised rates at any time based on events or by the remote operator. Figure 13 shows the Middleware commands that are sent through GPRS or SMS to dynamically configure, enable or reconfigure the GPIO's.

```
$PFAL,CNF.Set,AL14=IO.IN.e0=fedge:TCP.Client.Send,100,"INPUT1:HIGH"
$PFAL,CNF.Set,AL15=IO.IN.e0=fedge:TCP.Client.Send,100,"INPUT1:LOW"
$PFAL,CNF.Set,AL16=IO.IN.e1=fedge:TCP.Client.Send,100,"INPUT2:HIGH"
$PFAL,CNF.Set,AL17=IO.IN.e1=fedge:TCP.Client.Send,100,"INPUT2:LOW"
$PFAL,CNF.Set,AL18=IO.IN.e2=fedge:TCP.Client.Send,100,"INPUT3:HIGH"
$PFAL,CNF.Set,AL19=IO.IN.e2=fedge:TCP.Client.Send,100,"INPUT3:LOW"
$PFAL,CNF.Set,AL20=IO.IN.e3=fedge:TCP.Client.Send,100,"INPUT4:HIGH"
$PFAL,CNF.Set,AL21=IO.IN.e3=fedge:TCP.Client.Send,100,"INPUT4:LOW"
$PFAL,CNF.Set,AL22=IO.IN.e6=fedge:TCP.Client.Send,100,"BATTERY:LOW"
$PFAL,CNF.Set,AL23=IO.IN.e7=redge:GPS.Geofence.Park.Remove
$PFAL,CNF.Set,AL24=IO.IN.e7=fedge:GPS.Geofence.Park.Set
```

Fig. 13. GPIO's Enabling and Disabling Events "Over The Air"

The approach to the design of the Middleware Framework appears to be through the mapping of elements of the SWCD platform's feature model (see fig. 11) to the classes and packages. Atomic features have generally been turned into classes and composite features into packages. Additional packages and classes have been created for elements not covered by the hardware feature model but for which abstractions needed to be provided, such as utilities for handling concurrency etc. Classes have been interrelated using refinement/ abstraction, implementation, and generic dependency relations. Relations have been defined in such a fashion that hardware/OS dependencies have been kept within as few classes as possible, reducing the effort posed by adding support for a new platform. Common base and dedicated interface classes have been used to capture commonalities among devices of a kind, such as stream- or message-based communications devices/classes.

12. Geographic Information System (GIS)

A GIS system is a technique that manipulates, integrates and maps geographical information based on positional coordinates. (Latitude / Longitude). Many GIS software applications have been developed and integrated to precisely plot and display positional information, such as ArcGIS, MapPoint, Google Maps etc.

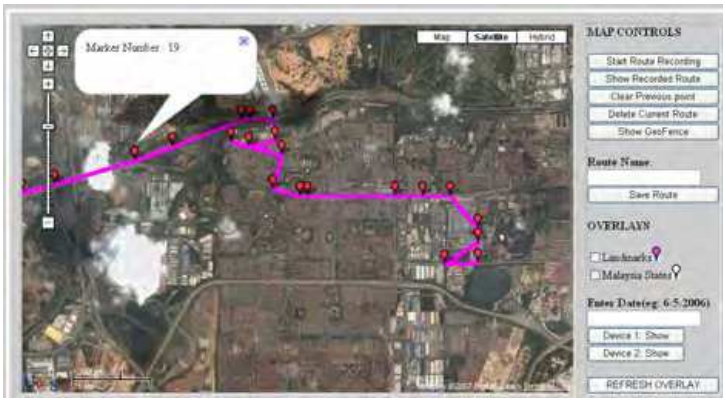

Fig. 14. GIS system used for Planning.

Companion papers (A.U. Alahakone 2007; Ragavan and Ganapathy 2007) describe the GIS systems that was developed along this work as a separate processes to remotely track the mobile object/robot on a web page embedding Google Maps APIs. Google APIs are freely available and accessible on the internet, and provide satellite maps as well as street maps. GIS systems incorporate several layers, each providing a different set of information to represent positional data. Google APIs provides the ability to embed many types of layers to enhance the quality of the data representation of the GIS system. This service as shown in Figure 14 is used to plan the path, create Geo-fence specifications and waypoint locations. Like wise options are available for the robot to be controlled and commanded over the web as a Tele-robot if need be.

13. Results and Discussion

In spite of a host of integration issues, software integration of the process modules was seamless. It has been established that dynamic heterogeneous systems can be evolved such as:- an embedded RTOS controller of the robot communicating through a GPRS mobile network, through a Windows based communication server that is a client/server application over TCP/IP, communicating the raw data acquired over a secure connection to be structured, processed, validated and fused at a remotely located server running on a different version of a Windows system across a Firewall, to be further stored in an Open Source Spatial MOD Database System running on MySQL, for further reporting and tracking of the robot movements in near Real-Time over a Web based GIS Tracking System continuously and accurately on a Map and send the information stored or created such as a Geofence or Route all the way back to the robot. The entire operation is completed with a round trip time of a few hundreds of milliseconds. Lastly it should be noted that our hybrid approach has considerably evolved over time based on lessons learnt in real-time and in distributed systems.

As a test example the Path Planning and Navigation System for Service Robots was successfully developed and deployed. It has also been established that Middleware can be used reliably to integrate disparate systems and processes and help in smooth evolution of Complex Dynamical Systems.

14. Conclusion

Modules and Components have been developed for an Asynchronous, Loosely Coupled to Uncoupled, Process Based, and Safe-Fail System as discussed and reported in this paper. The processes have been successfully deployed across hybrid and heterogeneous platforms from dedicated RTOS processors on the robot to distributed and disparate server machines connected through the World Wide Web. It has also been established that Middleware can be used reliably to integrate disparate systems and processes and helps in smooth evolution of Complex Dynamical Systems.

Having demonstrated how these strategies can be successfully implemented using the distributed networked software infrastructure such as Middleware, Webware and Hardware, a major challenge lies as future work in understanding how to make the most of it especially,

- understanding the tradeoffs between knowledge representations that are process based reactive, deliberative or hybrid and
- how to reduce the risk by managing software related failures in network controlled systems.
- Secure and fast methods for OTA download.

Acknowledgements

The authors wish to thank Monash University Sunway Campus for providing grants to purchase GPS modules, sensors, software and equipment necessary for the conduct of these experiments. The authors also wish to thank Transnoble Sdn Bhd, Kuala Lumpur for field support, streamed data, data acquisition and test communication server hosting.

15. References

A.U. Alahakone, S. V. Ragavan. (2007). Geographic Information System for Path Planning and Navigation of Mobile Objects. Conference on Innovative Technolgoies in Intelligent Systems & Industrial Applications, Kuala Lumpur , Malaysia. Kuala Lumpur , Malaysia. Proceedings of: Pages 6.

Arkin, R. C. (1990). "Integrating behavioral, perceptual, and world knowledge in reactive navigation." Robotics and Autonomous Systems 6(1-2): 105-22.

Arkin, R. C. and T. Balch (1997). "AuRA: principles and practice in review." Journal of Experimental and Theoretical Artificial Intelligence 9(2-3): 175-89.

Bagchi, S. and K. Kawamura (1992). An Architecture Of A Distributed Object-oriented Robotic System. Intelligent Robots and Systems, 1992., Proceedings of the 1992 lEEE/RSJ International Conference on.

Baliga, G., S. Graham, et al. (2004). Etherware: Domainware for wireless control networks, Vienna, Austria, IEEE Computer Society.

Baliga, G., S. Graham, et al. (2004). "Service continuity in networked control using etherware." IEEE Distributed Systems Online 5(9).

Brooks, R. A. (1991). Intelligence without reason, Sydney, Aust, Morgan Kaufmann Publ Inc.

Cattoni, R., G. Di Caro, et al. (1994). Bridging the gap between planning and reactivity: a layered architecture for autonomous indoor navigation, New York, NY, USA, IEEE.

Chochon, H. (1993). Object-oriented design of mobile robot control systems Springer Berlin/ Heidelberg.

Erann, G. (1998). Three-layer architectures. Artificial intelligence and mobile robots: case studies of successful robot systems, MIT Press: 195-210.

Gat, E. (1992). Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots. PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE, JOHN WILEY & SONS LTD. NUMBER 10, : 809

Hu, H., J. M. Brady, et al. (1995). LICAs: a modular architecture for intelligent control of mobile robots, Los Alamitos, CA, USA, IEEE Comput. Soc. Press.

Ibrahim, M. Y. and A. Fernandes (2004). Study on mobile robot navigation techniques. Industrial Technology, 2004. IEEE ICIT '04. 2004 IEEE International Conference on.

Kawamura, K., R. T. Pack, et al. (1995). Design philosophy for service robots. Systems, Man and Cybernetics, 1995. Intelligent Systems for the 21st Century., IEEE International Conference on.

Lewandowski, S. M. (1998). "Frameworks for Component-Based Client/Server Computing." ACM Computing Surveys 30(1): 3-27.

Lindstrom, M., A. Oreback, et al. (2000). BERRA: a research architecture for service robots, Piscataway, NJ, USA, IEEE.

Marco, G. (2003). A Flexible Object-Oriented Software Architecture for Smart Wireless Communication Devices. Proceedings of the conference on Design, Automation and Test in Europe: Designers' Forum - Volume 2, IEEE Computer Society.

Marcus, F., P. Wolfgang, et al. (2000). The UML Profile for Framework Architectures, Addison-Wesley Longman Publishing Co., Inc.

Miller, D. J. and R. C. Lennox (1990). An object-oriented environment for robot system architectures, Los Alamitos, CA, USA, IEEE Comput. Soc. Press.

Mohamed, N., J. Al-Jaroodi, et al. (2008). Middleware for Robotics: A Survey. Robotics, Automation and Mechatronics, 2008 IEEE Conference on.

Nesnas, I. A. D., A. Wright, et al. (2003). CLARAty and Challenges of Developing Interoperable Robotic Software, Las Vegas, NV, United states, Institute of Electrical and Electronics Engineers Inc.

Oreback, A. and H. I. Christensen (2003). "Evaluation of architectures for mobile robotics." Autonomous Robots 14(1): 33-49.

Ragavan, S. V. and V. Ganapathy (2007). A General Telematics Framework for Autonomous Service Robots. Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference on.

Ruangpayoongsak, N., H. Roth, et al. (2005). Mobile robots for search and rescue, Kobe, Japan, Institute of Electrical and Electronics Engineers Computer Society.

Schmidt, D. C. (2002). "Middleware for real-time and embedded systems." Communications of the ACM 45(6): 43-8.

Schraft, R. D. (1994). "Mechatronics and robotics for service applications." IEEE Robotics and; Automation Magazine 1(4): 31-5.

Simmons, R. G. (1994). "Structured control for autonomous robots." IEEE Transactions on Robotics and Automation 10(1): 34-43.

STEPPIII, F. (2009). "Falcom product website
http://www.falcom.de/fileadmin/downloads/documentation/STEPPIII/STEPPIII_flyer_v1.0.0_pre_web.pdf." last accessed 31.08.2009.

Urmson, C., R. Simmons, et al. (2003). A generic framework for robotic navigation. Aerospace Conference, 2003. Proceedings. 2003 IEEE.

Utz, H., S. Sablatnog, et al. (2002). "Miro - middleware for mobile robot applications." Robotics and Automation, IEEE Transactions on 18(4): 493-497.

Volpe, R., I. Nesnas, et al. (2001). The CLARAty architecture for robotic autonomy, Piscataway, NJ, USA, IEEE.

Zhang, L.-J. (2008). "Introduction to the IEEE transactions on services computing." IEEE Transactions on Services Computing **1**(1): 2-4.

Mobile Robots Navigation

Edited by Alejandra Barrera

ISBN 978-953-307-076-6
Hard cover, 666 pages
Publisher InTech
Published online 01, March, 2010
Published in print edition March, 2010

Mobile robots navigation includes different interrelated activities: (i) perception, as obtaining and interpreting sensory information; (ii) exploration, as the strategy that guides the robot to select the next direction to go; (iii) mapping, involving the construction of a spatial representation by using the sensory information perceived; (iv) localization, as the strategy to estimate the robot position within the spatial map; (v) path planning, as the strategy to find a path towards a goal location being optimal or not; and (vi) path execution, where motor actions are determined and adapted to environmental changes. The book addresses those activities by integrating results from the research work of several authors all over the world. Research cases are documented in 32 chapters organized within 7 categories next described.

INTECH
open science | open minds

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