Managing Resources for Better Behaved Robots

R. Kennelly, F. Morgan, M. Glavin
Dept. of Electronic Engineering, National University of Ireland, Galway,
Nun’s Island, Galway, Ireland.
ronan.kennelly@nuigalway.ie,
fearghal.morgan@nuigalway.ie,
martin.glavin@nuigalway.ie
http://www.ee.nuigalway.ie

Abstract
This paper presents an architecture based on features of the QNX real-time operating system (RTOS) that translate directly into aspects of a behaviour-based robotic model.

Significant research effort has been expended on the development of flexible frameworks to undertake new projects in the autonomous robotics field. It is argued that it may be more beneficial to tune a proven and flexible operating system to support this work rather than compromise on features to allow porting to other languages and environments.

The model presented is a reactive behaviour-based system that inherently boasts strengths such as modularity and robustness. The architecture is flexible enough however to allow different implementations such as Hybrid solutions to be quickly tested under realistic conditions.

The merits of the proposed approach are demonstrated by implementing an emergent wander behaviour using only small code segments. The behaviour allows a robot to travel in a human-centric environment at high speed and without collision.

The results presented in this paper have been obtained from a real robot in a cluttered office setting.

Keywords: Robotic architecture, behaviour-based robotics, QNX, Resource Manager, autonomous.

1. Introduction
Classical robotic research employed a deliberative architecture that used sensor data to create a symbolic representation of a robot’s environment. A robot would follow a sequential sense-plan-act loop that issued explicit instructions to achieve its objectives. System performance was poor as precise world models were impossible to obtain and planning decisions were too reliant on often-errorneous information from sensors.

The notion that useful tasks could be carried out by robots without precise instruction was first proposed by Brooks [1]. He suggested that the observation of an agent alone determined whether intelligence could be attributed to its behaviour. It did not have to comprise “intelligent” modules to exude intelligence as a whole. He maintained that agents could generate action by the correct organisation of simple structures working in unison and without a central world representation.

Brooks further claimed that “the world was its own best model” and that planning in real dynamic environments was a hindrance rather than a necessity. Accordingly, the general term for such arrangements became known as “reactive architectures”. Reactive architectures were later broken down further into “purely reactive” and “behavioural” architectures although this distinction is not often adhered to.

Hybrid architectures were later developed that contained both a deliberative element to plan long-term goals and a reactive layer to respond immediately to changes in the environment. It was found to be very difficult to achieve complex tasks through reaction alone. This model is now predominant in the literature.

Irrespective of the underlying architecture, modern robotic systems necessitate common fundamental requirements [2, 3]. This paper proposes that such requirements can already be met and exceeded by proven methods and that a framework should not be encumbered by the compromises demanded of portable systems. Brief examples of common aims include:

Loosely-coupled modules: Current research recognises the need to realise modules in isolation in order to thoroughly debug and test without having to take external dependencies into consideration. Designing loosely-coupled modules to run as distinct processes extends this idea further and promotes better fault tolerance. With separate processes, it is also easier to recognise irregular activity and revive problem modules automatically. Furthermore, distinct processes can be moved to other machines if they generate excessive processor load...
without disturbing the existing system. This guarantees architecture extensibility.

**Inter-process Communication (IPC):** Unless a system is designed around a monolithic process, a flexible and efficient IPC mechanism must exist to enable a coherent architectural structure. The IPC method cannot incur a significant overhead, nor should it have to repeatedly deliver data through a third party (i.e. processes should communicate directly with one another). It should also be sufficiently robust such that it is not a significant consideration when designing the main modules.

**Hardware abstraction:** Software often outlives the rapid progress in hardware technology. With this in mind, it is advisable to provide some means of hardware abstraction to shield algorithms from the exact source of information. More advanced sensors can then be easily integrated without requiring a major design effort. In addition, hardware abstraction provides for a complete hardware substitution, if desired. In practice this could mean replacing laser range information with sonar range measurements during a live run and without user intervention.

Subsequent sections of this paper illustrate how the QNX Real-Time Operating System (*RTOS*) satisfies the conditions above and provides additional features that could be very beneficial if converted into a robotic toolkit. The techniques and ideas presented here have all been tested on a Pioneer DX-3 robot in natural dynamic environments.

This paper is organised as follows: Section (2) outlines similar, on-going work in the area of framework development. Section (3) introduces the concept of the Resource Manager and its advantages. Section (4) discusses the specific Resource Manager written for the Pioneer DX-3 platform [4]. Section (5) summarises the hardware abstraction paradigm built over features of QNX. Section (6) details the design of the emergent wander behaviour written for the project.

2. **Related Work**

**Saphira [5]:** Saphira is a fuzzy-logic based development architecture that offers library packages for navigation, localisation and sensor interpretation. It also supplies a graphical simulator.

Saphira is based around the idea of a *Local Perceptual Space* (*LPS*). The LPS acts as a central knowledge repository for the system and holds information such as maps and sensor readings relevant to the various active behaviours. Behaviours are organised into a hierarchy and actuator commands are chosen based on their level of *desirability* according to fuzzy logic rules.

Saphira is a powerful tool, but developers are severely constrained by having to strictly follow a particular behavioural model and fuzzy logic rules.

**OROCOS [6]:** OROCOS is a European robotics initiative to provide a soft real-time toolkit for robotic development. It is based on the principles of hardware and communication abstraction, multiple contributing nodes and portability. OROCOS boasts its own special kernel for task execution and uses CORBA as its communication protocol. Lately, OROCOS appears to have moved away slightly from purely robotic projects and has become a more general control architecture.

**CLARAty [7]:** CLARAty was developed by the Jet Propulsion Laboratory (*JPL*) of NASA as a common architecture among its ever increasing fleet of planetary rovers. Its principal aim was to promote the reuse of code modules between different robotic hardware implementations to reduce future monetary and time costs.

The model consists of two layers, the *functional* layer and the *decision* layer. The functional layer consists mainly of reusable, extendible components that provide low to mid-range autonomy. The layer separates the algorithm goals from the base on which they are run. The decision layer is a deliberative-style arrangement that reasons about current system resources and attempts to reconcile them with current objectives. The decision and functional layers follow a client-server model that allows the decision layer to access the functional layer at various levels of granularity.

The code reuse and abstraction objectives of CLARAty are its greatest assets and its use on the Mars missions are an endorsement, but its tendency towards a deliberative nature may deter its adoption by others.

**Player/Stage** [8]: The Player/Stage design consists of Player, an interface to robot sensors and actuators and Stage, a 2-D simulation environment. A 3-D simulator named Gazebo has also been produced.

Player was designed as a network-based hardware interface using TCP to serve multiple clients. Its main design principle is that developers should have as much flexibility as possible in creating new robotic solutions while simultaneously removing the burden of writing hardware drivers and interfaces. Player also has the virtue of multiple language support through its choice of TCP as the communication protocol. Any language that recognises TCP can therefore be used for algorithm research, leaving Player to generate the appropriate low-level commands.

The structure of Player is most closely aligned with the intentions of the project work presented here. A significant difference between this paper and Player is Player’s insistence on the use of TCP. TCP provides the language independent functionality but induces significant overhead and can experience latency under a heavy load. These problems are not evident in the solution discussed in this paper which sacrifices multiple language support in favour of utilising trustworthy and well-tested components of a specific OS.

3. **The QNX Resource Manager Paradigm**

A QNX Resource Manager (*RM*) is a user-level process that clients can use to obtain access to a service but remain dissociated from the details of how the service is provided [9]. This service is similar to that provided by “device drivers” in other operating systems. In the case of QNX however, the process is external to the kernel. Implied in this is the fact that individual RMs can be stopped, updated and restarted “on the fly” without affecting the main OS, just like any ordinary application.
Network Transparency: RMs can be run transparently on any node in a network with no code modification through the QNX network manager qnet [10]. Unlike TCP or CORBA, message delivery between independent systems incurs only an extremely small penalty. Additionally, an RM can respond to command-line utilities such as echo, cat and ls, which can speed up testing and provides a means of easily checking what information clients are currently receiving or how hardware will respond to a particular command.

4. Robot Resource Manager

The Robot Resource Manager (RRM) written for this project as shown in figure 3, is a multi-purpose process responsible for all communication between the sensors and actuators.

This particular design was chosen based on the hardware composition of the Pioneer 3-DX robot used in this project. On this platform, sonar sensors, wheel encoders and motor functions are all handled by a Hitachi microcontroller through a serial connection (Microcontroller, figure 3) with an on-board PC. The RRM manifests the default QNX serial port driver to establish a connection to the microcontroller and follows its proprietary software protocol to maintain the connection. Other process threads are responsible for formatting received sensor and parameter data into convenient modules to service client requests (Packet Processing/Data structures, figure 3) and for maintaining “system health” (Signal Handlers, Watchdog Timers, figure 3). Client processes are not aware of the existence of the microcontroller or the serial connection. If all sensors (i.e. sonar, laser, camera, IR, etc) were implemented as discrete devices with custom APIs the client code to retrieve information would remain identical.

Specifically, the RRM in this project provides the following services for the Pioneer DX-3:

(i) Establish and maintain contact with the microcontroller
(ii) Packet reception and processing (from the microcontroller)
such as this has many beneficial side effects:

- Packet delivery to the microcontroller from external processes
- Error checking and error recovery
- Pathname registration and the timely updating of the information available at each pathname registered
- User display (screen) and direct user input (keyboard) management

The RRM paradigm imposes no restrictions on how a behaviour uses the data it supplies. The data available to all behaviours is always raw and unmodified, excluding error checking. A developer is therefore free to experiment with various set-ups, confident that results are not influenced by any external factors other than those under study. Behaviours written in this form are also highly portable because they are independent of the low-level hardware details on which they run and because they use POSIX calls. Transferring a routine to another system is a matter of starting the process on that machine (The target system is assumed to run an RM that has registered the same name as the original platform).

5. Behaviour-Based Robotics

As outlined above, the client-server design presented here poses no constraints on how data is used. This project however, has chosen to pursue the tenets of reactive-style architectures in parallel with the development of the framework. This section gives a brief overview of the main aspects of the reactive style realised as behaviours. Behaviour-based robots function by executing multiple simple tasks or “behaviours” in parallel. The behaviours share a common perception and compete for access to the actuators. Behavioural systems are judged on the complexity of their overall behaviour rather than the complexity of any individual contributor.

Multiple behaviours are structured in terms of “layers” that build upon the functionality generated by behaviours immediately below them. Internal communication between the layers, via wires, software channels etc, is largely non-existent in keeping with Brook’s idea of situatedness [11]. In practical terms, this concept states that behaviours should sense the environment themselves rather than receive a secondary interpretation from another layer or behaviour. Behaviours can communicate “through the environment” however, by watching for prompts triggered by other behaviours. A good example of such a case is in the architectural organisation of the can collecting robot Herbert [12]. In Herbert’s case a grasp behaviour is triggered by breaking an IR beam. This occurs when a behaviour responsible for the arm aligns the robot’s hand with a can. The arm behaviour does not send any internal signal to the grasp behaviour, but it conveys a message by breaking the beam of the IR sensor with the can (i.e. part of the environment). The independence of the grasp behaviour further allows a human to directly place a can in Herbert’s hand. The grasp reaction occurs as before because the IR beam is the only stimulus and does not depend on how it is triggered. Independent functionality such as this has many beneficial side effects:

- The interdependence of components is minimal and external so the robustness of the system is greatly increased. Localised failure can be tolerated, albeit resulting in reduced performance.
- Data corruption from sensors is less a consideration in the presence of multiple data streams. It is unlikely in a distributed system that more than one stream will suffer at any one time.
- This straightforward approach promotes robust code due to ease in writing and debugging components.

6. Wander Behaviour

In keeping with the stated aim of behaviour-based development and to demonstrate the ease of use of the RRM concept in practice, a wander behaviour, depicted in figure 4, has been developed. Only small code segments written in the C programming language were required to produce the routines shown using an RRM.

The wander behaviour is created by running two low-level behaviours, Steer Clear and Forward (figure 4), in parallel. The behaviours act as clients of the RRM and independently perform open() calls on the /dev/robot/ pathname to establish contact. Only information retrieved from the eight forward-facing sonar sensors is relevant to these particular processes. This sonar information is sufficient to avoid both static and dynamic obstacles but spurious readings can cause a collision if already very close to an obstacle. This could be overcome by adding redundant sensors such as IR and allowing clients to simultaneously read from that stream to rapidly cross-check readings before issuing a movement command.

The Forward behaviour designed for this project is merely the repetition of a forward translational command. It can be set to run at a predefined constant velocity or the parameter can be continually varied by the magnitude of incoming sonar readings.

The Steer Clear behaviour is written as an obstacle avoidance routine that rotates the robot in place until it faces an area free of obstruction. The algorithm is reliable enough to align the robot with openings as narrow as 50cm (robot diameter 38cm) while in motion. Steer Clear issues rotational commands based on the magnitude of the “force vector” perceived in a particular area. This model is a variation on the Potential Field Methods (PFMs) first proposed by Khalib [13], though only the fundamental idea that obstacles can be imagined to exert negative forces is utilised here. It was decided in the design of Steer Clear to shun vector summation to
avoid “false impressions” and the other problems that PFM’s suffer [14].

As an alternative to vector operations, sonar sensor data is grouped into “sectors” as shown in figure 5.

Figure 5: Sonar discs are grouped into sectors based on their angle

This arrangement recognises that obstacles appearing on the robot’s sensory periphery are less critical to robot safety than those that appear at the front. The sectors have a danger threshold set to reflect their position and the robot does not respond to readings that do not cross the threshold. In figure 5, Sector three (S3) is the most critical to robot safety while the threat is at its least serious in sector one (S1).

The decision to take sonar position into account rather than magnitude alone allows the robot to pass relatively closely to obstacles without them having an effect on its trajectory if they are not in the way.

Figure 6: An early version of the Steer Clear behaviour - displaying actual force magnitudes from a live run.

Figure 6 displays the readings from an early version of the algorithm. The length of the triangle emanating from the robot indicates the perceived magnitude of the force for that sonar sensor. The longer the triangle, the greater the force. The thick band shown is the robot radius, while the thin band (1000 approx, figure 6) is a threshold value above which the robot will take evasive action. Only a single common threshold appears in this diagram. 

Steer Clear uses a basic equation (1) to transform raw sonar readings into a figure that represents the magnitude of a virtual repulsive force acting on the robot.

\[ M = a - b \times \frac{\text{dist}}{T_s} \]  

(1)

where

\[ M \]  
Magnitude of the repulsive force

\[ a, b \]  
Positive constants

\[ \text{dist} \]  
Sonar distance reading to be converted

\[ T_s \]  
Threshold value for the sector in which the reading was received

This variation on Koren and Borenstein’s method [15] created for the project generates a large figure for objects detected close to the robot and falls to zero at the maximum range of the sonar sensors. The values of \( a \) and \( b \) are chosen so that

\[ a - b \times d_{\text{max}} = 0 \]  

(2)

where \( d_{\text{max}} \) is the maximum range of the sonar sensors. In this project \( d_{\text{max}} \) was chosen to be 3m.

The magnitude figure is normalised by the threshold value for the sector in which the reading was gathered, enabling general rules to be written for the behaviour. A magnitude greater than 1.0 is acted upon regardless of the sector in which it appears, while magnitudes below 1.0 can always be ignored.

When the rotation function receives a sector magnitude greater than 1.0, it issues a small rotation instruction (approximately 3°) to turn the robot away from that sector. It is possible to calculate and issue the exact rotation command required to guide the robot away from danger [16], but in practice this is not the best approach. A large steering command requires that subsequent packets be ignored until the turn is complete to avoid the command being re-issued. It also implies that the robot must track its position to know when to restart examining the packets.

In contrast, the Steer Clear behaviour issues small commands for the receipt of every packet indicating an obstacle. The robot manages to avoid a collision because commands are issued frequently enough and no positional information is required. There is an immediate two-fold benefit to this method:

(i) Spurious sonar readings will only cause minor fluctuations to the robot’s journey. This is important because sonar readings are noisy by nature and it is difficult to explicitly correct for this.

(ii) The obstacle avoidance method uses no positional data so encoders are not required.

Currently, packet reception occurs at a rate of 10Hz but there is a capacity to double this frequency by adjusting the parameters of the microcontroller. The present rate allows the robot a minimum velocity of 0.15 m/sec in situations so cluttered that openings are regularly just large enough for the robot to pass straight through. Furthermore, rapid left-right commands create an oscillation in extremely narrow openings that allow the robot to “wiggle” through a very small space.

Internally, command formation in Steer Clear is a logical series of “If-then” statements. The statements simply check for obstacles and turn towards open space.
If the intuitive command (i.e. right turn for obstacle on the left) would only lead to a further problem in the near future, the algorithm checks all sonar readings in an attempt to detect if the robot is headed towards an undesirable position. Summing the magnitudes of the forces to the left and the right of the robot in the presence of an obstacle will reveal if this is the case in most situations. If completely blocked by obstacles, the robot performs a small turning reverse away from the problem. Steer Clear also compiles other observations such as the number of free sectors at any particular instant and whether or not those free sectors are consecutive. These details may be used by another thread in the behaviour or offered via pathname registration to more competent behaviours.

7. Future Work
Standard device modules and algorithms using QNX could become a very powerful development tool. This paper has shown how features in QNX already solve problems faced by other frameworks. Future work will produce such standalone modules. Also, to compare this implementation more closely depends on the development of an effective behaviour arbitration scheme and an extension of the robot’s range of behaviours.

8. Conclusion
This paper presents an architecture that will allow modern robotic concepts to be quickly evaluated on real robots in unmodified settings, using common well-known interfaces for both process and hardware (POSIX). The system is modular, robust and fault tolerant. Features of the QNX RTOS are used to support the architecture. Two aspects in particular, memory protected user processes and resource managers, are very relevant to modern robust systems.

A wander behaviour has been created to show that effective obstacle avoidance with low processing requirements can be realised using this model. The wander behaviour emerges from a combination of two low-level behaviours that group sonar data into sectors and only react to obstacles if they exert a force greater than a predefined threshold for that sector. The solution allows the robot to pass very closely to obstacles without collision.

Acknowledgements
The authors would like to acknowledge the financial support of the Millennium Research Fund at the National University of Ireland, Galway.

References


