Complex AI on Small Embedded Systems: Humanoid Robotics using Mobile Phones

Jacky Baltes and **John Anderson**
Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, R3T 2N2 Canada
Email: jacky,andersj@cs.umanitoba.ca
http://aalab.cs.umanitoba.ca

Abstract

Until recent years, the development of real-world humanoid robotics applications has been hampered by a lack of available mobile computational power. Unlike wheeled platforms, which can reasonably easily be expected to carry a payload of computers and batteries, humanoid robots couple a need for complex control over many degrees of freedom with a form where any significant payload complicates the balancing and control problem itself. In the last few years, however, an significant number of options for embedded processing suitable for humanoid robots have appeared (e.g. miniaturized motherboards such as beagle boards), along with ever-smaller and more powerful battery technology. Part of the drive for these embedded hardware breakthroughs has been the increasing demand by consumers for more sophisticated mobile phone applications, and these modern devices now supply much in the way of sensor technology that is also potentially of use to roboticists (e.g. accelerometers, cameras, GPS). In this paper, we explore the use of modern mobile phones as a vehicle for the sophisticated AI necessary for autonomous humanoid robots.

Introduction

Until the last few years, intelligent mobile robotics has been greatly hampered by the size, power consumption, and computational limitations of available mobile computing platforms. While small mobile robots such as the Khepera have been used in applied research for many years, the applications of these were limited because of on-board processing ability, and the units were expensive (several thousand dollars each). More typical equipment likely to be encountered in the average AI lab would be platforms such as the Pioneer-II, which are large enough to carry laptops or full-size internal computing systems, but remain similarly expensive and carry significant demands due of their size (heavy lead-acid batteries and larger motors).

Conversely, recent years have brought about a revolution in available computational ability in embedded systems from the standpoint of mobile robotics. Smaller, powerful and less power-hungry processors, cheaper flash memory, and better battery technology have combined to allow far more effective embedded systems than were previously possible. Consequently, a generation of systems that are lighter and more robust now affords the possibility of smaller, lighter, and more adaptable robots. For the same reasons, these small, powerful embedded systems have also moved out of the industrial sector and into the realm of consumer electronics, giving much higher computational ability in embedded devices for everything from video equipment to automobiles. In particular, mobile phones have evolved from basic telephone and contact management abilities to handheld computers supporting sophisticated applications. The latter provide particularly exciting possibilities for AI and robotics: they combine powerful computational abilities with on-board peripheral devices (cameras, accelerometers, GPS units, bluetooth networking) that are in many cases improvements over what was available just a few years ago and would have to be separately managed.

Our work involves the development of control, planning, learning, and vision in humanoid robots. While small embedded processors have previously been used to power small humanoid robots (e.g. (Yamasaki et al. 2001), Manus I (Zhang et al. 2003), Tao-Pie-Pie (Baltes and Lam 2004), Roboerectus (Zhou and Yue 2004), and Hansa Ram (Kim et al. 2004)), these examples range in cost from $1000 to $20,000 US. Currently, we have moved from using these older types of embedded systems to developing sophisticated robotics platforms using mobile phones. Used modern mobile phones can be had for $100-$200 US (or indeed, even for free as a result of recycling programs) and provide all the facilities necessary to power complex adaptive humanoid robots for a fraction of the cost of several years ago.

Our interest in humanoid robots is in developing the kinds of broad adaptive behaviour that are necessary to support service robots of the future (e.g. for nursing or firefighting). These behaviours include being able to actively balance on uneven surfaces (e.g. move through grass or gravel), plan complex motions, such as crawling, carrying, and climbing, as well as combinations of these (e.g. pick up dirty laundry from underneath the bed), and interact with other robots or humans (e.g. move furniture in groups). The broad nature of these tasks is extremely challenging to AI in general, let alone intelligent systems running on small embedded processors such as mobile phones.

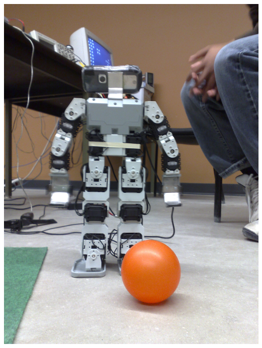

Figure 1: The modified Robotis Bioloid robot STORM

We have been competing for the last three years at major robotics competitions (RoboCup, FIRA) using humanoids whose main computational demands are supported using mobile phones. While RoboCup (RoboCup 2009) involves mainly soccer and a few challenges closely related to soccer (e.g. a ball throw-in), the FIRA HuroCup (FIRA 2009) competition is specifically designed to encourage the development of the types of broad robotic skills in which we are interested. The same physical robot must be able to participate in events ranging from basketball free-throws to obstacle course runs, to a climbing wall, taking place over extended periods of time. The computing demands to support the artificial intelligence necessary for such a range of activity (managing everything from computer vision, to active balancing and intelligent control, to localization and planning) would tax a full-sized desktop system, let alone a modern mobile phone.

This paper explores our experiences with using mobile phones for supporting sophisticated real time artificial intelligence in the domain of robotic control. We begin by describing our typical research platform. Following this, we describe with issues in adapting phones for these purposes, and discuss variations in OS, IO support, and issues in software development. We then illustrate the abilities of mobile phones for AI by describing three elements of our work that are representative of the difficulty of supporting AI on such systems: real-time computer vision, localization, and.

Humanoid Robots: Hardware and Software

For a physical platform, we begin with the Robotis Bioloid humanoid robot kit: these provide 18 degrees of freedom, use reasonably powerful and robust motors given their cost, and are far easier to acquire and assemble than building skeletal components from scratch. The Robotis kit includes a small AVR ATMega128 embedded controller for managing the individual servos of the robot. In our work, this is only used for low-level position control of the servo motors. Figure 1 shows one of our robots, STORM, using this platform, along with a mounted Nokia 5500 mobile phone for perception (visual feedback, active balancing sensors) and on-board computing.

The main drawback of using Mobile Phones is that they provide very little IO resources. We therefore add a custom-built IrDA interface, based on the Microchip MCP 2150 IrDA transceiver, to the humanoid kit. This allows the mobile phone to control the high-level motions of the robot.

While the Bioloid kit comes with firmware that can record and play back basic motions, this is not suitable for the complex motions we require, and so we replace this firmware with our own (requiring reverse-engineering part of the original Robotis firmware). The firmware also supports a 3-axis accelerometer from Analog devices, so that phones that do not have internal accelerometers can use an external sensor for active balancing.

Adapting Mobile Phones for Embedded Control Systems

There is is a huge variety of mobile phones available on the market, and dozens more are released each year. The cost of these devices is extremely competitive compared to many embedded systems (given their speed, memory, and included sensing devices), because they are produced in huge volume.

While economy of scale and the ability to have many necessary sensing devices included in an embedded system is very attractive to a researcher interested in supporting artificial intelligence and robotics on such systems, one is also well advised to heed the old motto: *Caveat Emptor*. Even from the same manufacturer, individual phones often have different versions of the same OS, support different extensions, and may sometimes run totally different OSs. The model number often confuses more than it helps in trying to decipher the OS that is run by a device. For example, the Nokia 6600 and 6680 are Nokia Series 60 devices, which is a very good OS for robotics purposes, whereas the Nokia 3300 and 3500 are Nokia Series 30 devices, which are not programmable. But the Nokia 6230 is a Series 30 device and the Nokia 3230 is a Series 60 device.

It is also important to realize that mobile phone manufacturers see these phones as finished consumer products, and therefore do not expect them to be "illicitly hacked" (from their perspective) to be used as embedded control systems. At best, some manufacturers encourage the development of third-party applications, but these applications often run in a sandbox which strictly limits which hardware is accessible to the application.

In spite of these hurdles, mobile phones can provide an extremely cheap development platform with high speed processing, LCD, buttons, wireless, bluetooth, infrared and one or two cameras in a very small and lightweight package. This section details our experiences with adapting these devices for robotics applications, including working with real time operating systems, developing software, and ultimately developing an IrDA interface for supporting IO.

A Tale of Caution

The most ubiquitous development environment for mobile devices is Java 2ME from Sun. It is available for a large number of devices and is standardized. However, J2ME really only standardizes the language and some of the GUI components, as well as data structures: several key technologies of interest to a researcher are only available as JNR libraries, which may or may not be supported.

For example, we purchased 13 "developer phones" from Sony Ericsson in 2004. These Z1010 phones include two cameras, Bluetooth, infrared, and had external storage on a memory stick. Initial development went smoothly and even though (as expected) the frame rate of the vision processing was slow, it would have been sufficient for robotic soccer. We found out the hard way, however, that Sony Ericsson does not allow access to Bluetooth nor IrDA infrared, nor to the external memory - even for developer devices. The company also refused to allow us to return the phones once we found out their limitations. We will therefore not recommend Sony Ericsson phones in any way.

Symbian OS Series 60

After several failed attempts trying to use J2ME for image processing, we chose the Nokia devices that run the Symbian OS S60 development environment. The main reason for this choice was that Nokia's SDK is more open and supports development in C++ and J2ME as well as other languages such as Python.

The Symbian SDK *Carbide* provided by Nokia for its phones is Windows-based and uses Eclipse as an integrated development environment (IDE). The tool chain includes the GCC Arm compiler, assembler, and linker, but also several other tools to help manage various builds (emulator, debug, release) and to help in internationalization. The first tool `bldmake` takes as input a `.bld` file which specifies the source files as well as the required libraries, and generates various makefiles in several directories and the `abld.bat` file which is used to control the build process. `abld` is a generated script that allows the building of debug or release versions for real hardware or emulators.

Even though Nokia only supports development under Windows and Eclipse, most of the Symbian tools are implemented in Perl and have been ported to Linux by the GnuPoc (GnuPoc 2009) project. This allows the development of S60 applications under Linux, which is our standard development method. The main drawback to this method is that most of the emulators do not run under Linux or the Linux emulation layer Wine. This means that all testing and debugging must be done on the phone hardware directly. Since most robotics applications are highly sensitive to timing issues, we found that the use of the emulator is not very useful in robotics in general.

The Symbian OS IrDA Interface

As previously mentioned, many phones possess an infrared port, which would be ideally suited to communicate with a small microcontroller responsible for various IO (e.g., accelerometers, gyroscopes).

There are three possibilities for achieving this, each with their own advantages and disadvantages.

Standard Serial Communication The conceptually simplest method is not to use the IrDA layer at all, but to use standard serial communication using infrared as the physical layer. This mode is supported on the early S60 phones, most notably the Nokia 6600. By loading the `ECUART` instead of the standard `IrCOMM` device library and by opening port `COMM::0` instead of `IrCOMM::0`, the phone will emulate a standard serial connection with IrDA multiplexing disabled. The advantage of this method is that any device that can be controlled using a standard serial line can be connected directly. For example, our humanoid robot DAUDANwas built using AI Motor serial RC servos, which use a standard 115 kbps serial communication protocol. Apart from the IR transceiver, no additional hardware was necessary. The significant disadvantage of this method is that Nokia has disabled this feature in newer S60 phones.

IrDA Interface Physical Layer only

Since newer models of the Nokia phones do not support UART style IR communication anymore, but only IrDA style communication, we investigated various methods for communicating between the phone and additional hardware such as microcontrollers. The interface consisted of a IrDA transceiver and IrDA demodulator circuit, which would take the IrDA infrared pulses and converts them into serial signals.

The problem is that the communication from the phone expects to establish a IrCOMM serial communication. This means that every message sent from the phone is wrapped in a message container with start- and end-of-message markers and checksums. However, in some cases the connected hardware (e.g., RC servo motors) uses a different start-of-message header and will simply ignore the additional container bytes.

Another version of our small humanoid robot, ABARENBOU was built using just the transceiver and the demodulator directly connected to AI Motor servos. The problem was that some servo positions when sent as an IrDA message would break the AI Motor firmware. In our case, these servo positions were not needed and we simply disallowed our application from sending these messages.

The main disadvantage of this method is that the phone can only send messages - it cannot receive them, since the AI Motor servos do not wrap their responses into a IrCOMM frame. So even though the servos provide position and torque feedback we were not able to use those in the motion planner.

IrDA Interface IrCOMM Layer in Hardware

The IrDA protocol provides a serial port emulation layer called IrCOMM, which emulates a full serial port including flow control and handshaking signals. It also automatically multiplexes various channels over the IrDA link. The IrCOMM layer is built on top of the IrDA physical layer.

To implement a full IrDA layer on a small microcontroller is a non-trivial task. Therefore, many hardware designers use the Microchip MCP-2150 IrDA protocol controller, a small chip that provides IrCOMM layer communication without any additional hardware of software.

Our experiences with the Microchip MCP 2150 were mixed. The MCP 2150 has some bugs, which ultimately means that it is not able to establish a IrCOMM link with the Linux IrDA stack. Furthermore, devices based on S60 20 and 21 can communicate with an MCP 2150, but the newer S60 30 and 31 cannot establish a link.

Since the MCP 2150 wraps the communication into a IrCOMM frame, the phone can both write and read messages

```
<State id="Scan For Target" > <Enter>
%%v(angle) = 0;
if ( previousState == %%State("Target Right Forward") ) {
      %%v(newAngle) = 20; /* Turn 20 degrees first */
      %%v(angleAdjust) = +10; }
 else {
      %%v(newAngle) = - 20; /* Turn 20 degrees first */
      %%v(angleAdjust) = -10; }
</Enter>
<Process>
  if ( ( %%v(newAngle) >= -190 ) && ( %%v(newAngle) <= 190 ) ) {
      if  (%%v(angle) != %%v(newAngle) )  {
           turn( (%%v(angleAdjust) * TEN_DEGREE) / 10 );
           %%v(angle) = %%v(angle) + %%v(angleAdjust); }
      else {
           %%v(newAngle) = - %%v(newAngle) - 40;
           %%v(angleAdjust) = - %%v(angleAdjust); } }
  else {
      %%Transition("Random Walk"); }
</Process>
</State>
```

Table 1: An XML schema for a behaviour that scans for a target by turning right/left with increasing sweeps

from the attached hardware. We used this scheme in our small humanoid robots STORM (figure 1), and ROGUE.

IrDA Interface RCOMM Layer in Software

In light of these experiences, we suggest the use of the IrDA protocol with the physical layer only as the most flexible solution. We implemented our own minimal IrCOMM layer to establish a connection between the phone and our hardware. Unfortunately, this requires the use of a microcontroller (e.g., PIC or AtMega AVR). However, many designs for complex robots such as humanoids or snake-like robots already require a microcontroller for motion planning and processing of sensory information.

Complex AI on Mobile Phones

Having described some of the hurdles that have to be navigated in terms of employing mobile phones as robot control hardware, we now discuss the actual use of these platforms for sophisticated, real-time artificial intelligence. There are many significant and interacting AI problems in the domain of mobile robotics, which is why this domain is ubiquitous from the standpoint of both research and teaching.

As a basis for dealing with a complex world, a robot must first be able to move about in it coherently, and combine reactive and deliberative reasoning. Our agent architectures are reactive and behaviour-based, and use behaviour trees to support a balance between deliberative planning and reaction. A behaviour tree involves successive levels establishing a context for lower-level behaviours, which are implemented as finite state machines. These are described using our own XML-based meta-language, in order to provide a machine-processable description of the intent of a behaviour. The specification of behaviours includes preconditions (enter functions) and postconditions (exit functions). A slightly simplified example of a simple behaviour that scans for a target with increasing sweeps is shown in Table 1.

The XML schemas include additional markup to refer to states by name (`%%State("Random Walk")` access variables (`%%v`) and to trigger transitions to other states (`%%Transition`).

Behaviours are organized into behaviour trees. Higher

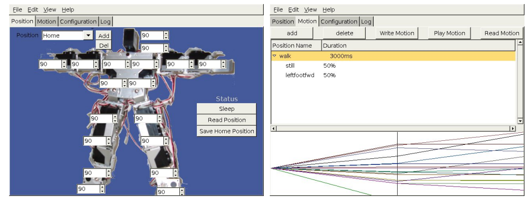

Figure 3: Interface of our motion development system. Left, the development of a position; Right, the combination of these positions into motions.

level behaviours can override or enable other lower level behaviours. For example, a *Perception* behaviour may disable the scan for target behaviour and enable the state *Target In Front* if it recognizes the target.

One of the design goals of the meta language was to be highly efficient. Instead of adding a XML parser and interpreter to the agent, the meta language is parsed and interpreted offline and converted into highly efficient C code. This code is then compiled and executed on the mobile phone. For example, the example above shows that the programmer uses state names (e.g., "Random Walk," and "Scan For Target"). However, the states' names are converted to integers in the C code. Because of this formalized state representation, we can also easily generate alternative representations when they are useful, such as visualizing the finite state machine as a graph. For example, figure 2 shows the state transition graph for a simple approach task. The robot first approaches a target and then walks away from it.

The actual behavior code on the mobile phone must ultimately prescribe movements for the robot. These movements are defined atomically, and are developed beforehand as motor control programs using a software interface (figure 3. The interface allows one to move the robot into a specific position and save this position. The interface also allows one to set the trim (i.e., offset) for all joints as well as the home position.

A separate window tab is used to combine these positions into motions. Each motion has a cycle time associated with it and each part of a motion has a arrival time associated with it. Thus, the interface allows the user to easily adjust the speed of a whole motion or individual parts of the motion. The trajectory of all joints is shown in the bottom window. STORM (figure 1) has twenty atomic motions, including: *start walking*, *take step with right foot*, *take step with left foot*, *stop from left walk*, *stop from right walk*, *sideways step left*, *sideways step right*, and *kick with right foot*.

These movements are then available to be played back as required by any of our programs running on the mobile phone - here, by the particular state out of a behaviour currently in control of the robot. In order to respond adaptively within a given state, and appropriately make transitions between states, a robot must be able to perceive its current environment (real-time vision), and know its current location within it (localization and mapping). The remainder of the paper explores the design and implementation of these two facilities, using mobile phones, as examples of adapting so-

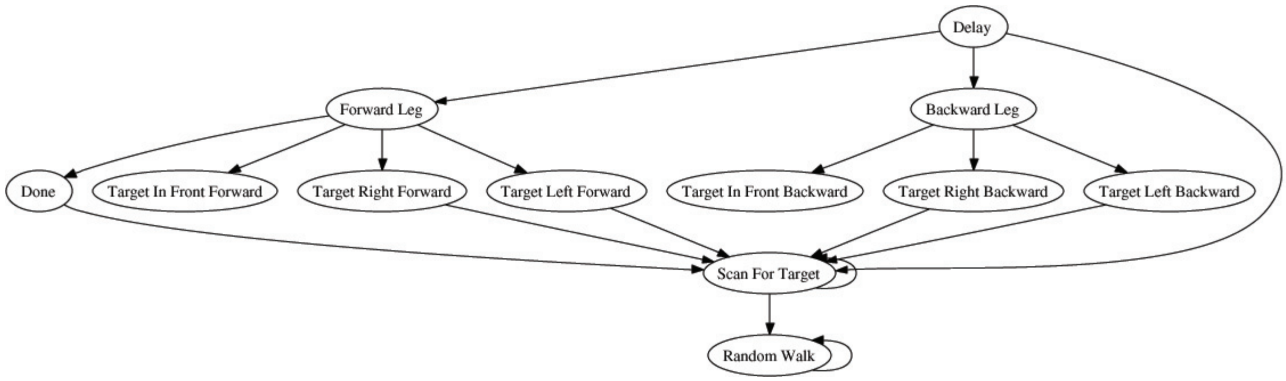

Figure 2: Automatically generated state transition graph for a simple approach and avoid task. Solid lines are state transitions.

phisticated artificial intelligence to these embedded devices.

Vision Processing in Robotic Soccer

STORM uses the Nokia's camera as its main sensor. The camera is used to properly approach objects in the field of view of the robot as well as to supply information for localization and mapping.

To be robust enough to deal with a complex environment such as robotic soccer, the vision processing makes little use of colours, and makes use of a very fast approximate region segmentation algorithm. First, the algorithm scans the image and extracts scan line segments (i.e., segments of similar colour) of approximately the right size. This step is similar to standard region segmentation algorithms.

However, we noticed that implementing a full union-find algorithm was too slow for a mobile phone, since it took about 2 seconds per image. The adaptations needed here are typical of adapting sophisticated computation to mobile phones: since most objects of interest in the soccer, environment are relatively small, we use a flood fill pixel merge algorithm, to find the associated region for a scanline. The flood fill algorithm keeps track of which pixels have previously been visited, and thus will visit each pixel at most once. The returned region is then checked for size (i.e., number of connected pixels), size of the bounding box, aspect ratio, and compactness. Only in the final step does the algorithm test whether the average colour of the region matches the object colour. If any of these tests fail, the object is rejected. Using only average colours of regions results in robust recognition of the ball and the goals and takes on average approximately 200ms.

An approximation of the relative position of objects is possible by determining the pan and tilt angles of the phone (from the servo on which it is mounted), and then calculating the distance to the centre of the image. In this domain, it is safe to assume that these objects are on the ground plane. The relative position of an object at the centre of the image will have the closest approximation, so the camera is centered on important objects such as the ball before a decision is made as to what action to take next.

Goals are also detected as objects. Each goal is a distinct colour (prescribed by RoboCup rules). If both goal colours are found in one image, the regions of each goal colour are merged with other regions of the same goal colour. The goal colour that is present in the largest merged region is considered to be the goal currently being viewed.

To help the feature-based localization method described in the next section, we use a complex camera calibration based on the Tsai camera calibration algorithm (Tsai 1986). This calibration is only done once for each robot. Given this calibration information, we are able to map points in the image accurately to their real world coordinates. This is essential, because it allows us to determine the distance and orientation of the ball to a feature point (ball, goal post, line)

Before localization can occur, features must be extracted from the image. The relevant features for localization on the soccer field are lines, goals, and the centre circle. Every 5th column in the image, the system scans from the bottom of the image towards the top. If there is a transition from a green pixel to a white pixel, the pixel p is remembered in a list. The scan continues upward, so there may be more than one transition pixel in a column.

Lines are then found by running a gradient guided Hough transform (Hough 1962). For each point p_i, a set of adjacent points is determined. Triplets are formed from these by including one point to the left of the point p_i, and one point to the right of p_i. There are several triplets that can be formed this way out of the neighborhood of adjacent points. Each triplet votes for an unbounded line in the image. This vote is fuzzified by voting for a small range of slopes through the point p_i.

The peaks in the Hough accumulator space determine the equations of possible lines. For each peak in the accumulator space, we search along the pixels determined by the line equation to find start and end points of the lines. This results in a set of line segments.

The line segments are ordered based on their size. The longest line segment is assumed to represent the edge of the playing field. Given the distance and gradient of the line segment, the position and direction of the robot can be computed.

Note that the above approach is biased toward particular types of objects (the ball, goals) and is heavily reliant on lines. One of the more forgiving aspects of robotic soccer

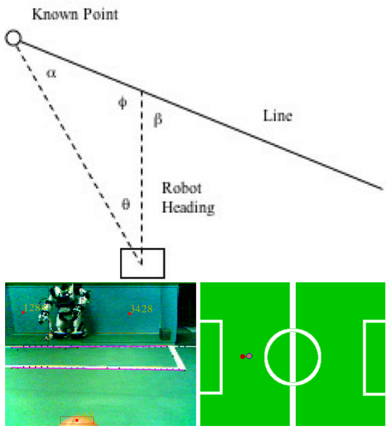

Figure 4: Localizing using a known point, its relative position, and relative orientation of a line. Image from the robot with highlighted line segments and the calculated position are shown in the lower image.

is that the playing area is extremely structured in terms of lines compared to most common, everyday environments. Elements of many domains (e.g. driving uses lines extensively) can be similarly approached, but we are still a long way away from performing general vision using embedded systems such as mobile phones.

Localization and Mapping in Soccer and Beyond

Just as most complex domains require sophisticated sensing such as vision for feedback, most domains require knowing at least something about one's position in order to act appropriately. Moreover, most domains will allow performance in general (including the ability to better know one's own position) if an up-to-date map of the world around the agent is available. In some domains, reasoning can be specific enough to be able to focus on certain aspects of the domain in this regard. In soccer, for example, knowing the position of the ball is especially important. The ball's position relative to the robot is easily determined from an image containing the ball. However, without knowing the world position of the ball, we cannot do anything useful with it: kicking in just any direction could put the ball out of bounds or even into the robot's own goal. Even in a basic case such as this, the problem of localization and keeping at least a limited map of what is occurring on the field is thus crucial.

Absolute localization of the robot plus relative localization of the ball will give absolute localization of the ball. Absolute localization can be calculated as long as a point is viewed with a known world coordinate, and knowing the robot's world bearing from it. One instance of this in soccer is when a goal post is seen (figure 4). Once this is accomplished, dead reckoning can be used with some accuracy for a short time afterward.

Because soccer is a fast-moving game, this is the extent of localization performed by STORM. Just as previously occurred with vision, one of the factors being relied upon here is the significant structure inherent in a soccer field. Complex domains that humans deal with easily, such as navigating a previously-unknown room in an average home, generally have much less structure that can be relied upon. Moreover, such domains may require a longer-term map compared to moving on a soccer field.

There are many approaches to doing more sophisticated localization and mapping. One that particularly showcases the use of the mobile phones we employ is multi-agent simultaneous localization and mapping (SLAM) - beginning in an unknown environment, a group of robots must create a shared map while navigating using the partial map created thus far in order to explore further. There have been numerous prior approaches to this problem (e.g. (Burgard et al. 2000; Rekleitis, Dudek, and Milios 1997)). In our work (Bagot, Anderson, and Baltes 2008), we use multiple robots navigating in an obstacle course environment, where obstacles are colored to allow vision to be the sole sense employed. Figure 5 shows STORM and ROGUE, both using Nokia mobile phones, jointly navigating using this method. Each robot maintains its own map, but shares information via the phone's bluetooth to the other robots employed, so that mutual perspectives constrain interpretation and correct error. Based on this shared information, they also make sensible choices for choosing frontiers to explore in the unmapped portions of the domain. While much prior work on SLAM exists, most of this is done with wheeled robots using sophisticated sensors such as laser rangefinders. In our work, using vision alone is more error prone: the motion and jitter introduced by humanoid movement both makes ongoing tracking of objects more challenging and introduces greater error in estimating the robot's pose.

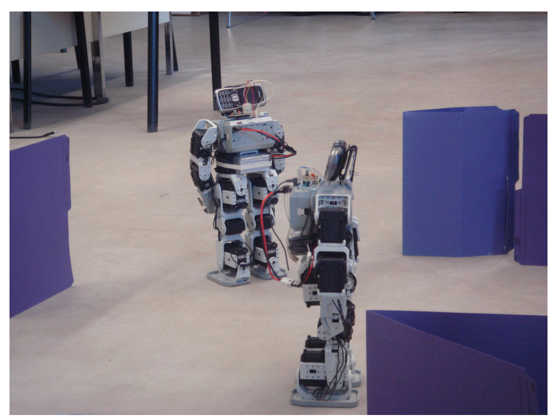

Figure 5: STORM and ROGUE jointly performing SLAM on an obstacle course.

A SLAM solution gradually builds a map by mapping visible spatial area relative to the current estimated pose of an agent, which is unknown to begin with. Therefore any odometry error during motion propagates to landmark location in the map. In our work, we employ a particle filter, based on that of Rekleitis (Rekleitis 2003). A particle filter is a probabilistic model that maintains a number of weighted estimates of the robot's current pose (location and orientation) as particles, and uses new incoming evidence to filter out incompatible hypotheses and propose new ones. The particle population size is 100, which is manageable with the mobile phones' processing power, but successful results have been reported (Kwok and Fox 2004) with a particle population size of 50. Population depletion is handled with a simple select with replacement re-sampling algorithm, as used by Rekleitis (Rekleitis 2003).

Our particle filter differs from that of Rekleitis (Rekleitis 2003) in its motion model (wheeled robots were used in (Rekleitis 2003), whereas humanoids must account for more error-prone motion and also movements that are impossible for wheeled robots, such as a side step) and in its particle weight update method. After an action (e.g. a left, right, forward, or backward rotation or translation) the pose estimate of each particle is updated based on the motion model (i.e. an estimate of how far a step or turn should be able change the robot's position), and then the weights are updated based on visual feedback. The pose estimation with dead reckoning is much more difficult to deal with in this work than Rekleitis', because we are using a single camera whose view is altered by the many degrees of freedom of the robot, rather than a fixed laser scanner. While the pose estimation is based on the motion model, the camera's visual feedback is used to alter the weight, to avoid the accumulation of estimated odometry error that would otherwise occur. Our image processing approach returns the polar coordinates of objects in the camera's field of view, but during locomotion this is very noisy due to motion blur. Our weight update method consequently uses a certainty factor in the camera data and a constant decay. The best particle at any point is then the weighted average of all particles.

In this work, landmarks are mapped relative to the best particle in an occupancy grid (with 25x25cm grid cells) with a recency value associated with each grid cell. A recency value $[0, 255]$ is associated with each grid cell instead of the more common posterior probability. If the recency value of a grid cell is greater than zero, a landmark is believed to exist in the corresponding grid cell. This recency value is updated based on current perception. If the sensor senses an object, and the coordinates of the object relative to the best particle in the particle filter map to a grid cell with a recency value greater than zero, then the recency value is incremented; otherwise, it is initialized to 128. If the sensor does not sense an object, landmarks are extended to circles with radius r, if a line segment with length l (maximum sensor range) extended from the best particle intersects a landmark circle, the recency of the corresponding grid cell is decremented (Figure 6).

Each agent communicates its estimated pose and all landmarks in its local map, along with its target pose (desired coordinates and orientation for motion planning) to other agents using the bluetooth capabilities of the mobile phone. Agents attempt to reconcile landmarks in their own local maps, possibly extending these, and also store the pose and target pose of the agent nearest to them.

To support a global coordinate system, we adopt the sequential deployment technique of (Anderson and Papanikolopoulos 2007). Agents enter the environment one after another from the same pose, which results in the same unique origin in a local coordinate system for each agent. Thus, when describing the location of a landmark, no rotation or translation information is required. The weakness of this is the error in local pose estimation, but that itself should be improved over time as SLAM unfolds. Internally, each agent maintains the pose of the nearest agent and their current target pose.

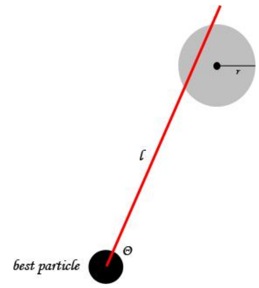

Figure 6: Recency update method.

Unlike (Burgard et al. 2000), we use no centralized method for dealing with frontier selection. We also do not maintain a shared set of targets requiring consistent communication from all agents. Instead, we attempt to use implicit coordination to minimize coverage overlap between agents without relying on communication, by exploiting the fact that each agent maintains the communicated pose and target pose of the closest agent to it. As SLAM unfolds, agents select target poses such that each must be a frontier on its local map, but in addition, the Euclidean distance from the target pose to the nearest agent must also be greater than the maximum sensor range.

In our approach, the sharing of information is designed to improve the accuracy of SLAM but still allow it to operate without multiple individuals. That is, should bluetooth communications be interrupted, each agent's own particle filters will still operate using their own perceptions: they will simply not have as broad a range of information on which to base decisions about the best particle. Similarly, agents will still make the best decisions they can for deciding where to explore, but with a poorer estimation of where the agent nearest it is going, there will be more redundant exploration performed.

The evaluation presented in (Bagot, Anderson, and Baltes 2008) did not show appreciable improvement in mapping accuracy by using the input of multiple agents. We believe this was because of the sheer volume of error in this domain (i.e. there are many similar landmarks) along with error in movement that could not be accounted for because we did not employ an internal model of the agents' walking gait, but even as it stands the model should be useful in a domain where landmarks are more predictable. The work does show, however, that something as sophisticated as a particle filter taking in real-time vision and the communicated perceptions of other agents is possible to successfully deploy using mobile phones.

Discussion

In this paper, we have described our experiences with adapting mobile phones for use in artificial intelligence and robotics, including dealing with operating systems and overcoming restrictions on IO facilities. We have also described work specifically in robotic control software development for mobile phones, in computer vision, and in performing localization on these devices.

While there are still hurdles to overcome in using mobile phones broadly in AI, there are some promising signs for the future. In 2008, the Symbian Foundation announced their plans to merge software assets from various manufacturers to form **Symbian Platform**, a open source version of the Symbian OS released under the Eclipse Public Licence. In late 2009, the first version, **Symbian^1**, was released. The Beagleboard (BeagleBoard 2009) was the first hardware that was targeted by this version.

Nokia has also been pushing Posix compatibility by providing a posix emulation layer for Symbian OS and acquiring Trolltech, the company that makes the excellent QT graphical user interface library.

Therefore, we believe that Symbian OS and Nokia phones in particular will provide even better opportunities for embedded control applications in the future.

In the mean time, the fact that there is a natural tension in manufacturers between wanting to see these devices used while trying to ensure that commercial control remains intact means that those wanting to use these as general computing devices will regularly see obstacles placed in their paths. The best bet in navigating these obstacles is a large group of technically-able and dedicated users.

Just as one example, with version 31 of the Symbian OS, Nokia introduced a mandatory signing process for applications. For applications that wanted to make use of certain features - most notably to us, wireless bluetooth - the file system had to be digitally signed before being uploaded to the phone.

Initially, a developer was able to request (free of charge) a signing key that was valid for a limited period of time for a specific phone. In 2008, the process changed and an application had to be uploaded and signed via a website, which required a working Internet connection during development. While this is only a small restriction, it caused a big problem for us, since during robotics competitions, Internet access is sketchy at best. Luckily, hacks for various phones disabling the signature check or the capabilities check quickly emerged and allowed us to use the phones during competition.

Disclosure

The autonomous agents laboratory at the University of Manitoba received five Nokia N95/N96 devices in 2008 through the Nokia University Relations program.

References

Anderson, M., and Papanikolopoulos, N. 2007. Improving multirobot, cooperative search via local target queues. In *Proceedings of IEEE IROS-07*, 2590–2595.

Bagot, J.; Anderson, J.; and Baltes, J. 2008. Vision-based multi-agent slam for humanoid robots. In *Proceedings of the 5th International Conference on Computational Intelligence, Robotics and Autonomous Systems (CIRAS-2008)*, 171–176.

Baltes, J., and Lam, P. 2004. Design of walking gaits for tao-pie-pie, a small humanoid robot. *Advanced Robotics* 18(7):713–716.

BeagleBoard. 2009. Beagleboard website. http://www.beagleboard.org.

Burgard, W.; Fox, D.; Moors, M.; Simmons, R.; and Thrun, S. 2000. Collaborative multi-robot exploration. In *Proceedings IEEE ICRA-00*, 476–481.

FIRA. 2009. Fira hurocup rules. http://www.fira2009.org/gamerules/hurocup-laws.pdf.

GnuPoc. 2009. Gnupoc website. http://gnupoc.sourceforge.net/.

Hough, K. Y. 1962. Mehod and means for recognizing complex patterns. U.S. Patent 3069654.

Kim, J.-H.; Kim, D.-H.; Kim, Y.-J.; Park, K.-H.; Park, J.-H.; Moon, C.-K.; Ryu, J.-H.; Seow, K. T.; and Koh, K.-C. 2004. Humanoid robot hansaram: Recent progress and developments. *JACIII* 8(1):45–55.

Kwok, C., and Fox, D. 2004. Map-based multiple model tracking of a moving object. In *Proceedings of RoboCup-2004*, 18–33.

Rekleitis, I.; Dudek, G.; and Milios, E. E. 1997. Multi-robot exploration of an unknown environment, efficiently reducing the odometry error. In *Proceedings of IJCAI-97*, 1340–1346.

Rekleitis, I. 2003. *Cooperative Localization and Multi-Robot Exploration*. Ph.D. Dissertation, McGill University.

RoboCup. 2009. Robocup website. http://www.robocup.org.

Tsai, R. Y. 1986. An efficient and accurate camera calibration technique for 3d machine vision. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 364–374.

Yamasaki, F.; Matsui, T.; Miyashita, T.; ; and Kitano, H. 2001. Pino the humanoid: A basic architecture. In Stone, P.; Balch, T.; and Kraetszchmar, G., eds., *RoboCup-2000: Robot Soccer World Cup IV*. Berlin: Springer Verlag. 269–278.

Zhang, R.; Vadakkepat, P.; Chew, C.-M.; and Janardhanan, J. 2003. Mechanical design and control system configuration of a humanoid robot. In *Proc. of 2nd Int. Conf. on Computational Intelligence, Robotics and Autonomous Systems (CIRAS 2003)*.

Zhou, C., and Yue, P. K. 2004. Robo-erectus: a low-cost autonomous humanoid soccer robot. *Advanced Robotics* 18(7):717–720.