ABSTRACT
The recent drive towards ubiquitous computing has been universal. We strive to make devices more intelligent and to embed these devices deeper and closer into our lives. For a device to behave in a manner that we perceive to be intelligent it needs to be extremely context sensitive. Thus, our measure of intelligence usually depends on how much the device adapts to its environment. The most basic environment that we can change for an “intelligent” device is its location. Let’s consider the example of a simple device which tells a hiker which direction the rest of his group is. Obviously if the device loses its functionality as soon as the hiker places a few steps in any one direction, the device wouldn’t be of any use to him. Likewise a personal digital assistant that gives information that is localized to the owner’s current location would be perceived far more intelligent than a device which does not know where it is in the environment. This paper addresses a system which will calculate the locations of nodes on a wireless network. Combined with work by Johnny Yeh on the localization computation layer and Reino Virrankoski on condition checking the complete system will be able to localize nodes successfully. The portion of the system that this paper addresses is the measurement protocol, the hardware abstraction layer and the interface layer. The measurement protocol provides collision free measurements and passes it up via the interface layer. The localization computation layer does the final computation of locations.

1. INTRODUCTION
There are many benefits to localization. In an increasingly connected world, small devices which are capable of locating themselves in the environment have a critical role to play. This ranges from a direct role like that of a PDA to functionality which embeds itself and extends the functions of a different system. For example a PDA could use location information to give directions to a user. Another possible use might be in self organizing networks where for example robotic nodes could reorganize themselves to optimize communication bandwidth usage. There are many applications where if a node’s location relative to other nodes is known it would be very useful. Examples can be found in military applications as well, where mine locations relative to each other or controlling formations can be done if localization information is known.

2. PROBLEM DESCRIPTION
The Localization problem is to calculate the positions of nodes given distance measurements between pairs of nodes which are able to communicate with each other. When the exact locations of at least 3 nodes are known in a rigid 2-dimensional environment, it is possible to pinpoint exact coordinates for all the nodes in that rigid region. However if exact coordinates are not known (no beacon nodes) it is possible to create a relative coordinate system whereby each node would know the position of other nodes relative to its own position which provides most of the advantages that absolute localization does.

A challenge that is addressed in this project is to implement the localization system in an ad-hoc way, where all the nodes have the exact same program running on them and figure out distances between them in ad-hoc. They are not synchronized by a central node. The previous research involved a central node synchronizing the measurements between each pair of nodes. Collisions among nodes must be avoided and resolved when they do occur.

3. SYSTEM OVERVIEW
The complete system can be shown in a block diagram as in figure 1.

![Figure 1](image)

Obtaining the measurements in ad-hoc and passing them over to rigidity checking and clustering is done by the measurement protocol. The rigidity checking and
clustering module works on the measurements to identify rigid regions of the graph. Each rigid region is split into clusters with an identified head-node. The head node could run either collaborative multi-lateration or Multi-dimensional Scaling (MDS) on each cluster to figure out guesses for each node’s position, which are then used in the refinement phase to obtain better coordinates.

The system was implemented as a stack of layers each of which works on the output generated by the previous layer. The four main layers starting at the top are the Computation and Refinement Layer, Rigidity Checking and Clustering Layer, Interface Layer, and the Measurement Protocol Layer.

**Figure 2**

### 3.1 COMPUTATION & REFINEMENT LAYER
This layer involves the senior project done by Johnny Yeh. The head node that was determined by the clustering & rigidity checking is the only node that functions in this layer. The other (non-head) nodes do not pass information up to this layer and their only function is to communicate and measure distances. The input to this layer is a distance table containing a node’s 2-hop neighborhood. The basic functionality of the layer is separated into three parts, verification, computation and refinement.

#### 3.1.1 VERIFICATION
This portion basically checks the distance table it received for values that are obviously incorrect. For example, distances that are too large to be accurate or measurements that seem inconsistent are removed and where possible repaired based on the other values in the distance table.

The layer also functions as an outlier detector. It removes edges of the graph that were formed because of obstacles and echos which make the distance seem larger than the actual distance.

#### 3.1.2 COMPUTATION
First the layer runs the Floyd-Warshall all-nodes shortest paths algorithm to figure out the shortest path distance between each pair of nodes. These distances are not going to be completely accurate and the error increases with the number of hops between each pair of nodes. The computation layer searches through the distance table to find the three nodes that create the triangle with the largest surface area between them. The triangle with the largest surface area is used because this allows more nodes to be localized since there is higher probability of a node being within range of the highest area triangle. The three nodes that make up this triangle form the axis for the newly formed local coordinate system. These three nodes are used a beacon nodes to localize the remaining nodes.

Trilateration is used on each non-beacon node with the three beacon nodes to find an initial estimate of the node’s location. Since the location of a point is completely defined if we know the distances from 3 other points to that point this is straightforward. However, the co-ordinates we obtain have large errors in them depending on the connectivity of the initial graph. If the initial graph (before Floyd Warshall) was fully connected then the errors are only the errors introduced in measurement. But if a large number of the distances are found over many hops using Floyd-Warshall, then the co-ordinates that are found will be inaccurate. However, since we use them only as initial estimates for the next phase this is not a problem.

#### 3.1.3 REFINEMENT
The refinement portion of the computation layer performs the critical task of aligning the node positions obtained, to the distances that were used in obtaining them. This stage greatly increases the accuracy of the system. The algorithm used is a Kalman Filter which uses the Least Square Minimization method to refine the values.

### 3.2 CLUSTERING & RIGIDITY CHECKING
This layer is part of ongoing work by Reino Virrankosky. It has not been integrated into the project work yet, but will be integrated as soon as work on it completes. The layer needs initial estimates of node positions which it generates from the 2-hop distance tables. The first part consists of running the pebble game by Bruce Hendrickson on the data and determining which areas of the graph are rigid. For localization to succeed a unique realization of the graph must exist. The requirement for this is for the graph to be 3-connected and redundantly rigid. The pebble game can be used to obtain these redundantly rigid regions. There isn’t much that can be done on non-rigid regions...
because there are multiple possible locations for those nodes. If maximizing the performance over multiple runs is the requirement, we could make a guess as to the location of the non-rigid (and non-3-connected) nodes based on the information that we have of the node. Alternatively if measurements need to be more accurate we can discard these nodes since they increase the errors in the upper layers (Especially if the MDS algorithm is used for calculation).

The second part is to run a clustering algorithm (Currently under development by Reino) to separate the graph into clusters, each with a head-node which is the node that will continue the next level of computation. Each cluster will be a separate local co-ordinate system which will be constructed by the Computation layer. The aims of the clustering algorithm are to break the graph into clusters in such a way that each node in the cluster can still be localized (i.e. we do not reduce the amount of useful information we have), and the clusters can be recombined to create a complete co-ordinate system which is consistent.

3.3 INTERFACE & MEASUREMENT LAYERS

This paper mainly addresses these two layers. Although the behavior is explained in detail below I will first summarize the role of each layer.

The interface layer is the link between the measurements obtained from the Medusa node and the OKI ARM processor. It listens for measurement packets from the UART and updates a global distance table depending on the values of the packet. If there were changes, it sends an event to the layer immediately above it which makes it update its computation. Currently it’s the computation layer that is above the interface layer but once the condition checking layer is complete the event will be sent to that layer.

The measurement layer is a protocol for calculating the distances between nodes and it runs on the Medusa MK-2 node. The protocol which will be detailed below finds the 2-hop distances between nodes and also detects and recovers from collisions. The communication between the measurement and interface layers is via the UART on each node. The packet format is described in detail in this paper.

4. SIMULATION

A simulation without actual nodes was done using the MDS method for the initial estimates. Since the simulation could be done on a computer with much more processing power than the nodes, the simulation was also done in a centralized fashion.

The simulation system had five components:
1. Run the pebble game to identify redundantly rigid regions
2. Floyd Warshall algorithm to obtain Shortest paths to all nodes
3. MDS algorithm run on this matrix giving initial estimates for node positions
4. These positions used as initial guesses for a Kalman Filter (Least squares Minimization)
5. Convert positions to absolute positions if there are at least 3 beacons in each region

The system was tested at scale with the number of nodes ranging from 5 to 100 and with different degrees of connectivity. This simulation provided extremely good results (shown at the end of the document) and showed that condition checking and then computing relative co-ordinates is a system that works very well.

For step 5, we used a 2*2 matrix with four unknowns as the rotation matrix and a 2*1 matrix with two unknowns as the translation matrix and were able to find the transformation which converts from any relative co-ordinate system to an absolute co-ordinate system given the location of 3 beacon nodes.

5. ENVIRONMENT

The localization system would be run on a system of hybrid nodes. Each node consists of one Medusa MK-2 node and an ml67q4003 board. The ML67Q4003 board was powered by an OKI ARM (ML67Q4003) processor. The new node designed at Yale University by Prof. Savvides will be used in the future. The Medusa MK-2 node (figure 3) is itself a hybrid node consisting of an AVR (ATMega128L) processor and a AT91FR4081 ARM/Thumb Processor. We only use the AVR portion of this node.

Figure 3

The reason for combining the Medusa Node was to make use of the ultrasonic transceivers that were built on the node. These are shown in Figure 4. The node has 8 ultrasonic transceivers which form a half dome of coverage around the node. Since the coverage is half a dome the node is ideal for placing on the floor or on the ceiling to cover a full area. Both nodes are equipped for serial communications via the onboard UART. The interface cable that was used only needed three of the pins on the UART.
The operating system on both parts of the hybrid node consisted of ports of PALOS (Power Aware Lightweight Operating System).

6. PALOS

Palos is an example of an operating system for an embedded device. As these devices are largely constrained by resources and processing power, the development paradigm for building OS’s have been to customize the operating system for each application. Therefore none of the parts that are compiled into the final binary that runs on the hardware are redundant or unnecessary. The application itself is a customization of the operating system. This allows developers to use the limited resources effectively with minimal development effort. The first part of the project involved porting PALOS to run on the ML67Q4003 node. This was done in three steps. The first involved porting the PALOS core, which is the part of the operating system which is essential for the running of even the most basic application. The second step involved porting the timer and stopwatch modules which are the basic modules behind applications that depend on timing. The third step was to port the UART driver which was used to drive communication between the OKI Node and the Medusa MK-2 node.

6.1 PALOS CORE (palosTask.c and palosMain.c)

To allow the Palos Core to function, the first step involved changing the Hardware Abstraction Layer (HAL) so that the registers and hardware addresses that Palos used would correspond to the correct addresses on the ML67Q4003 nodes. The abstraction of the hardware related calls to a separate layer made the porting simpler as values could be changed in one location to change the complete system. An issue that we had here was that there was no exact one to one mapping between the registers on the AVR and the registers and their functionalities on the ARM processor. The basic structure of the Palos core can be shown as:

1. Initialize
2. Repeat steps 3-4
3. If there is an event in the event queue fetch it
4. Call the correct task with the event as a parameter

This provides an extremely intuitive and simple mechanism for tasks running on PALOS to interact. First every task that requires to be evoked registers itself as an event handler of a particular type. Whenever one task needs to send an event it does this simply by creating an event and handing it over to Palos with the task ID of the task to deliver to. The stopwatch module allows events to be sent based on a time constraint, where a task can wake up after a fixed amount of time.

6.2 STOPWATCH AND TIMERS (stopwatch.c & timer0)

The functionality of the stopwatch module is to be the scheduler for PALOS and it depends on using one of the hardware timers available on the processor. Therefore in porting from AVR to ARM this was one aspect that needed changing to use the correct timer interrupt. When the timer calls stopwatch it keeps a count of how many ticks have passed since the last call for each task that has requested the stopwatch functionality. Whenever the number of ticks passed is larger than the amount of time that the task requested, an event is created for that task. Although this only provides an approximate timing, for the project involved, it was found to be adequate.

6.3 UART

The UART was the last part of the system to be ported. Several changes were made to the function of the UART device driver. The way that the driver on the AVR was functioning, was to send out character after character but there was no way of figuring out if one character had been sent successfully before sending out the next one. The AVR UART driver simply assumed that the characters would come in at a slow enough pace where the chance of overwriting was relatively small. The ARM on the other hand had an interrupt which flagged that the hardware send buffer was empty. By overriding this interrupt it was possible to send enough characters to fill the send buffer and then when the system informed that the send buffer was empty (via interrupt), to send more characters. This system was much more robust and removed possible errors from overflowing the hardware send buffer. Another change that was done in the UART driver was to fix the previously unused readNbytes() function. This allowed a task to send multiple queries for a fixed number of bytes. The system would send the task an event with the necessary data when the bytes became available over the serial port. This allows functionality such as requesting 5 packets of data of different sizes where each packet would be sent to a different task as long as they arrive in a predefined order.
6.4 TIME GUARANTEES
PALOS does not preempt any task that takes longer than its allocated time to run. This means that a misbehaving task could go into a loop completely taking up the processor. However with the way that the operating system is designed together with the application to take-up minimum overhead, we can assume that all tasks will hand back control in a reasonable amount of time. PALOS is not designed to provide RTOS (Real Time Operating System) like hard time guarantees and only provides a best effort (soft) time guarantee.

6.5 PORTING ISSUES
An issue that arose in the porting process was a perceived weakness of the arm-elf-gcc compiler. Unlike the corresponding compiler for avr, it does not support selective compiling of functions that are required by the program, but instead compiles the entire library even if only one function is being used from it. With the restrictions on memory usage on the processor, this was a big disadvantage. Initially the work-around was to use a minimum number of libraries but eventually the code was compiled to run on the flash memory of the device which had higher capacity and could still function even if extra libraries were compiled. Ideally, this would be something that is fixed in a future version of the compiler.

7. THE MEASUREMENT PROTOCOL
(ahlmeasure.c)
The measurement protocol was designed to run in a purely ad-hoc fashion. As the protocol was run on the lower power Medusa MK-2 node, the algorithm needed to do a minimum amount of processing. The algorithm also needed to avoid collisions between nodes as there was no longer an arbiter node synchronizing the measurements.

The layers involved in the protocol can be graphically shown as in figure 5.

![Diagram](image)

The protocol can best be described by a state machine.

7.1 INITIAL STATE
When a node is first switched on, the node will do initializations, create the network discovery packet which will be the first packet it sends, and then go into a waiting state where, the node will wait for a random amount of time. During this waiting state it can receive packets from other nodes but will not send packets. The node marks itself as clear to send but will not send until the timer expires. When the node’s timer expires the node will enter the network discovery state.

7.2 WAITING STATE
If a node receives a broadcast network discovery packet in this state, if this is the first time it received a Network Discovery packet from this node while in the current working state, it will subtract 1 from clear to send which is initially set to 1. Any value less than 1 means that the node is waiting for a different node to complete transmission and will stop the node from starting any transmission. The node will reset a “Not clear to send” timer and the node will automatically assume that it is clear to send once this timer expires.

If this was not the first time it saw a packet from this node, the node will take measurements as normal [This means that the initial measurement done packet has been dropped and the node has sent a second network discovery packet]. The measurement process is explained below under “Measurement process.”

If instead of a network discovery packet, a measurement done packet is received in the waiting state, if clear to send is less than 1 (only time the node would have clear to send set, but still come to this point is if the initial network discovery packet was lost), the node will increment clear to send and if clear to send is 1 will also stop the “Not clear to send” timer it started earlier. This basically means that all pending measurements have completed so the node is now clear to send. If clear to send is still less than 1 it means that there are still pending measurements happening (with other nodes) therefore the node should still be in the not clear to send state. In receiving the measurement done packet the node will also complete the measurement process and create a packet to be sent across the UART to the OKI ARM node. This packet contains the latest measurement obtained as well as the list of neighbor distances for that neighbor.

Another possibility that could happen in the waiting state is for the “Not clear to send” timer to expire. If this happens it means that the node will assume that enough time has passed for the neighboring node to complete the measurement, therefore it is assumed that the measurement done packet must have been dropped. Therefore the clear to send is set back to
and the node will wait in the waiting state until its back off timer expires before entering the network discovery state again.

### 7.3 NETWORK DISCOVERY STATE

In this state the node will send out a network discovery packet, which basically contains its own ID and a list of its neighbor distances. The packet also functions as a signal to all the other nodes around that the medium is busy and that they should not start a network discovery. The node will also increase the back off timer up to a maximum of 15 seconds, and set it again. Also the node will set up a timer which will repeatedly send out ultrasonic pulses from each of the 3 transmitters on the perimeter of the node and will also repeat each transmission 3 times. The repeated transmissions are to average out noise and to provide more accurate distance measurements. The reason for using multiple transmitters on the perimeter is to find the shortest distance between the two nodes. Because of the way sound travels if the transmitters are facing different directions the transceivers could get readings which are inaccurate because of echo’s etc. Therefore the smallest reading is assumed to be the correct one since it represents the shortest path.

After the ultra-sound transceiver has sent out the final packet that consists of one measurement it will wait for the maximum amount of time that the last packet could have taken to arrive at a node. We assume that nodes are not separated by more than 5 meters. Therefore the maximum amount of time it takes is \((5/330 \text{ s})\). A timer is set for this amount time and when the timer expires the node will transmit the measurement done packet. This signals to all nodes around the packet that the measurement is complete and the medium is now free again.

---

**Figure 6** – State machine showing the measurement protocol

**Figure 7** – Packet format
### 7.4 MEASUREMENT PROCESS
Each measurement consists of an ultrasound packet and a radio packet. The time difference between receiving the radio packet and receiving the ultrasound packet gives a measure of how far two nodes are from each other.

When the measurement done packet is received the receiving node takes the smallest of the measurements received (which corresponds to the shortest path distance) and also averages the measurements received from each ultrasound transceiver to improve the accuracy.

### 7.5 COLLISION AVOIDANCE & DETECTION
Because the first network discovery packet functions like an RTS for the Media Access Control (MAC) protocol, other nodes refrain from sending while a measurement is under way. Collisions are avoided via this mechanism. The range of the radio packets tends to be larger than that of the ultrasound packets; therefore the packets tend to stop communication in a wider area than is absolutely required. Therefore, the chance of collision is even less likely.

Collisions can occur however via the well known hidden node problem. Where two nodes that are on opposite sides of a receiving node transmit and neither transmitting node receives the other’s network discovery packet therefore it is not aware that the measurement was interrupted. With our system however measurements happen repeatedly every 15 seconds, and therefore a single measurement missed does not cause a problem. However the code attempts to use heuristics to identify instances where this type of collision has occurred and the packets have been corrupted.

### 7.6 COMMUNICATION BETWEEN NODES
When a new measurement is obtained, the measurement protocol running on the Medusa MK-2 node, sends a packet via the UART to the OKI ARM ML67Q4003 Board. This packet has the following information: Node ID, Neighbor Node ID, Distance to that neighbor (newly obtained) and the list of distances to the neighbor’s neighbors (neighbor’s neighbor list – 2 hop neighborhood).

### 8. INTERFACE LAYER

<table>
<thead>
<tr>
<th>Node ID</th>
<th>Neighbor’s ID</th>
<th>Distance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Distance from neighbor to node 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Distance from neighbor to node 2,…</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Distance from neighbor to node n</td>
</tr>
</tbody>
</table>

**Figure 8 – packet format between AVR and OKI**

On the ML67Q4003, the Interface layer handles the incoming communication from the UART.

The functionality of the interface layer is very straightforward. For each packet it receives (Figure 8) it will add the entries in the packet to the global distance table matrix. This matrix is accessible by both the interface layer as well as the layer immediately above it. After updating the matrix, the interface layer creates an event for the parent layer. This event simply has one ID field which says that it’s a new update event. This event is then queued for the parent layer.

### 9. EXPERIENCES & CONCLUSION
The project gave many insights on working and designing programs to run on multiple platforms. The experience cross compiling code to run on the AVR and the ARM nodes was very interesting. A good understanding of the hardware was necessary to program the measurement protocol and the guidance provided by Prof. Savvides was invaluable.

The issue with the ARM compiler including the complete library when a single function call is used added a challenging aspect to the project. It made it necessary for the program to be run on ROM. The configuration changes that were required, as well as the steps involved in debugging the program running in ROM, provided many learning opportunities.

The results obtained from the measurement protocol showed that the measurements were being obtained in a collision free and efficient manner. The transfer to the OKI Board via the UART and the interface layer functionality happens smoothly and packets are transferred. When Reino’s condition checking capabilities are complete, it needs to be integrated to the system so that a cohesive system is
in place. Even as it is, although it does not do condition checking, the system provides a localization system that performs extremely well compared to previous methods at localization.

10. FUTURE WORK
Instead of using a hybrid node to use the ultrasonic transceivers on the AVR node, a single node should be designed which has the required sensors to do distance measurement. A compiler which does not include a whole library for a single function call is required, which was a limitation that the project had to work with. If some sort of signature can be transmitted by the ultra sonic sensor rather than anonymous pulses it would be able to detect collisions much more reliably. The impact of this project on devices of much smaller scale developed in the near future will be immense.
REFERENCES


[ii] Location Service Protocol Stack for Wireless Networks of small devices (Senior Project) – Johnny Yeh, 2004

[iii] “Distributed Geometry Aware Ad-hoc Clustering,” Reino Virrankoski, Yale University, ENALAB, 2004


[viii] Andreas Savvides, Embedded Networks and Application Laboratory, Yale University, 2004