ABSTRACT
The freshwater pond snail *Lymnaea palustris* is an easily maintained organism that can be used in the developmental biology laboratory. *L. palustris* is also among the gastropod molluscs who exhibit capsular development. The capsule in which the embryo develops provides nutrients and protects it from contamination and other harmful external factors. If the capsule is damaged or removed, the developing embryo is at higher risk for abnormal development or mortality. Although the capsule provides many benefits for the embryo, it causes interference to manipulation for developmental studies and experimentation. By studying the components and properties of the capsule an artificial capsular fluid (ACF) can be produced. The production of this ACF could facilitate survival of the embryo without the capsule, allowing for manipulation of the embryo. This study seeks to establish *L. palustris* as a developmental model system for the teaching lab. To that end, we have also modified or developed techniques for DNA staining and immunohistochemistry. These techniques will help to further develop the utility of *L. palustris* as a model in developmental biology.

KEYWORDS: *Lymnaea*, embryonic development, capsular development

1. INTRODUCTION
*Lymnaea palustris* is a freshwater snail frequently found in aquatic vegetation such as ponds, rivers, and streams, preferring neutral, calcium-rich water. *L. palustris* can be identified by the shape of its shell, which exhibits a dextrally coiled asymmetrical shape once fully developed. These snails exhibit hermaphroditic reproduction, conducting fertilization internally via self- or cross-fertilization [1]. The fertilized eggs in capsules are oviposited in transparent jelly masses which adhere to the substrate or tank. Within two to three hours after oviposition, cell division begins. Meiosis generates two polar bodies before the first mitotic cell division takes place. Observation of the polar bodies can aid in identifying the animal pole. Prior to cleavage central spindle formation takes place between two pronuclei, initiating cytokinesis [2]. The eggs then enter early cleavage which displays equal cleaving up to the four cell stage. After the four cell stage, the spiral pattern emerges and asynchronous cleavage in a 2, 4, 8, 12, 16, 20, 24 cell pattern [1]. The eggs divide approximately every 70 minutes until the 24 cell stage. The 24 cell stage is a resting period which lasts up to three hours before cleaving again. Gastrulation occurs about 16 hours after the first cleavage. The vitelline membrane ruptures at the vegetal end of the embryo; as
the invagination of the vegetal pole deepens, the animal pole begins to flatten. This gives the gastrula a placode shape. When an embryo is undergoing gastrulation, it can be observed spinning, a result of the formation of cilia on the embryo. Although the rotation is barely visible in gastrula embryos because it is so slow, it can be much better observed by the trophophore stage.

The trophophore stage occurs two days after oviposition. In this stage the embryo begins to develop kidney and shell glands. On day three, the embryo enters the veliger stage. This stage is critical to development because the shell begins to form, the foot becomes apparent, and the embryo beings to elongate. Four to five days into development the eyes become visible. The foot is now visibly separate from the head region and the embryo has somewhat of a hippopotamus appearance; this stage is cleverly nicknamed the “hippo” stage. By the time the embryo has reached this stage it is enlarging and taking up more and more volume within the capsule. The embryo will soon develop into a juvenile snail (7-8 days) when its size will cause it to burst out of the capsule and jelly mass. The juvenile snail will then obtain nutrients from the vegetation and grow to an adult reaching sexual maturity within six to eight weeks [2,3].

*L. palustris* is an ideal specimen to use as a model for developmental studies due to its easy acquisition and maintenance in the laboratory and its rapid development. However, traditional developmental laboratory manuals focus on such organisms as chick, sea urchin, mouse, *Xenopus*, and *Drosophila* and there has been less focus on the snail in widely used resources [4]. More recent research has served to lead the way for molluscs to be included as developmental models, notably the mud snail *Ilyanassa* [5]. A number of different developmental concepts (internal fertilization, spiral holoblastic cleavage, protostome gastrulation, cell-cell communication) and techniques (embryo culture, microsurgery, histology, microscopy) can be illustrated using the snail embryo [2]. Thus, with further research and development on this organism we hope to expand its suitability for developmental biology experimentation and design a suite of techniques useful with *L. palustris*.

The goal of this study was to produce an artificial capsular fluid (ACF) that can support survival of embryos without their capsule. In order to test the ACF, other techniques such as observation of landmark developmental stages, clean and precise decapsulation (to avoid damage to embryos), and sterile techniques are needed. Additionally, in order to increase the usefulness of this organism as a developmental model system, its suitability for techniques such as fluorescent immunohistochemistry was investigated.

## 2. MATERIALS AND METHODS

### 2.1. Observing and decapsulating embryos

*L. palustris* are kept in filtered tanks of artificial pond water (APW; 0.45 g Instant Ocean salt, 0.2 g calcium sulfate and 0.3 g sodium bicarbonate per liter of RO water). As needed snails are fed organic romaine lettuce. To obtain embryos, oviposited jelly masses are detached from the walls and base of the tank and transferred to APW for observation and staging [1] on a dissecting microscope.

A scalpel blade is used to slice open the jelly mass so embryos can be rolled out onto filter paper then transferred to a dish of APW. Embryos may now develop normally in capsules, or be subject to decapsulation (under the dissecting scope with two pairs of sharp forceps). As each capsule is popped open the embryo is quickly transferred to a new dish of APW or ACF with a glass pipette. Named stages correlate with days post-fertilization as is indicated in Table 1.
Table 1. *L. palustris* stages

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Day 0 post-fert.</td>
<td>Cleavage</td>
</tr>
<tr>
<td>Day 1 post-fert.</td>
<td>Gastrula</td>
</tr>
<tr>
<td>Day 2 post-fert.</td>
<td>Trochophore</td>
</tr>
<tr>
<td>Day 3 post-fert.</td>
<td>Veliger</td>
</tr>
<tr>
<td>Day 4 post-fert.</td>
<td>Hippo</td>
</tr>
<tr>
<td>Day 5+ post-fert.</td>
<td>Juvenile snail</td>
</tr>
<tr>
<td>Day 7-8 post-fert.</td>
<td>Hatch from capsule</td>
</tr>
</tbody>
</table>

2.2. Formulation of artificial capsular fluid

Embryos were decapsulated and placed in APW alone or with enhancement. In one trial 3% Ficoll (Sigma F4375) was added to APW to increase osmotic pressure; other supplements are described below. Capsulated embryos as controls were placed in all treatments. Embryos were cultured for 48+ hours and observed for viability.

Composition of supplemental salts for ACF was obtained from previous work on capsular fluid. Four cations were measured in natural capsular fluid, listed in order of importance to embryo survival: calcium, magnesium, sodium, and potassium [6]. We combined these with albumin and sugars to mimic the natural capsular fluid.

In the preliminary attempt to formulate ACF 0.3g glucose, 0.3g galactose, and 0.2g bovine serum albumin (BSA) were added to 50 mL APW. It failed to increase survivability, likely since the capsule contains almost double the salt of APW [6]. Thus, a mixture of salts was added to RO water along with BSA, glucose, and galactose (Table 2).

Table 2. ACF1 composition

<table>
<thead>
<tr>
<th>Component</th>
<th>Concentration</th>
</tr>
</thead>
<tbody>
<tr>
<td>KCl</td>
<td>.04 g</td>
</tr>
<tr>
<td>CaCl</td>
<td>.21 g</td>
</tr>
<tr>
<td>MgCl</td>
<td>.05 g</td>
</tr>
<tr>
<td>MgSO₄</td>
<td>.05 g</td>
</tr>
<tr>
<td>NaHCO₃</td>
<td>.03 g</td>
</tr>
<tr>
<td>NaCl</td>
<td>.03 g</td>
</tr>
<tr>
<td>BSA</td>
<td>.2 g</td>
</tr>
<tr>
<td>Glucose</td>
<td>.3 g</td>
</tr>
<tr>
<td>Galactose</td>
<td>.3 g</td>
</tr>
</tbody>
</table>

The production of ACF was repeated several times making adjustments as necessary. Some of the other trials included the following components and amounts (Table 3-5).

Table 3. ACF2 composition

<table>
<thead>
<tr>
<th>Component</th>
<th>Concentration</th>
</tr>
</thead>
<tbody>
<tr>
<td>KCl</td>
<td>.04 g</td>
</tr>
<tr>
<td>CaCl</td>
<td>.21 g</td>
</tr>
<tr>
<td>MgCl</td>
<td>.05 g</td>
</tr>
<tr>
<td>MgSO₄</td>
<td>.04 g</td>
</tr>
<tr>
<td>NaHCO₃</td>
<td>.03 g</td>
</tr>
<tr>
<td>NaCl</td>
<td>.03 g</td>
</tr>
<tr>
<td>BSA</td>
<td>.2 g</td>
</tr>
<tr>
<td>Glucose</td>
<td>.3 g</td>
</tr>
<tr>
<td>Galactose</td>
<td>.3 g</td>
</tr>
</tbody>
</table>

Table 4. ACF3 composition

<table>
<thead>
<tr>
<th>Component</th>
<th>Concentration</th>
</tr>
</thead>
<tbody>
<tr>
<td>KCl</td>
<td>.04 g</td>
</tr>
<tr>
<td>CaCl</td>
<td>.21 g</td>
</tr>
<tr>
<td>MgCl</td>
<td>.04 g</td>
</tr>
<tr>
<td>MgSO₄</td>
<td>.05 g</td>
</tr>
<tr>
<td>NaHCO₃</td>
<td>.03 g</td>
</tr>
<tr>
<td>NaCl</td>
<td>.03 g</td>
</tr>
<tr>
<td>BSA</td>
<td>.4 g</td>
</tr>
<tr>
<td>Glucose</td>
<td>.3 g</td>
</tr>
<tr>
<td>Galactose</td>
<td>.3 g</td>
</tr>
</tbody>
</table>

Table 5. ACF4 composition

<table>
<thead>
<tr>
<th>Component</th>
<th>Concentration</th>
</tr>
</thead>
<tbody>
<tr>
<td>KCl</td>
<td>.04 g</td>
</tr>
<tr>
<td>CaCl</td>
<td>.35 g</td>
</tr>
<tr>
<td>MgCl</td>
<td>.05 g</td>
</tr>
<tr>
<td>MgSO₄</td>
<td>.05 g</td>
</tr>
<tr>
<td>NaHCO₃</td>
<td>.03 g</td>
</tr>
<tr>
<td>NaCl</td>
<td>.03 g</td>
</tr>
<tr>
<td>BSA</td>
<td>.6 g</td>
</tr>
<tr>
<td>Glucose</td>
<td>.3 g</td>
</tr>
<tr>
<td>Galactose</td>
<td>.3 g</td>
</tr>
</tbody>
</table>

The mixtures were placed on a nutator to gently mix and dissolve without shaking. The ACF was sterile filtered at 0.45µm porosity. ACF was refrigerated when not in use.
2.3. Immunohistochemistry

Decapsulated embryos were fixed in MEMFA (MOPS/EGTA/Magnesium sulfate buffer with 3.7% Formaldehyde) at 4°C for 24 hours; MEMFA was replaced with PBS (Phosphate-buffered saline; 2X 15 minutes). Embryos were transferred to PBT (PBS + 0.05% Tween-20), followed by blocking (10% goat serum in PBT) for 1 hour at room temperature. Embryos were incubated in primary antibody, 1:100 rabbit polyclonal anti-tubulin (Santa Cruz Biotechnology sc-5546) or p-PKC α (sc-12356-R) in PBT, at 4°C 24 hours. Embryos were washed in PBT 3X 30 minutes, 1X overnight at 4°C, and 2X 30 minutes. The secondary antibody, 1:100 mouse anti-rabbit IgG-FITC (sc-53805) in PBT, was applied at room temperature 4-6 hours. Embryos were washed with PBS and observed under epifluorescence.

To minimize loss of embryos during the numerous changes of solutions, we made “basket-buckets” for the embryos by cutting off the bottoms of 1.5ml tubes and melting nitex mesh of 90µm porosity on the bottom. Since the embryos are approximately 100µm in diameter, they were contained in the basket-buckets for all solution changes by transfer between wells of a 24 well plate. These basket-buckets aided greatly in the processing of *L. palustris* embryos (Figure 1).

![Figure 1. Side view (A) and top view (B) of basket-buckets.](image)

DAPI (Sigma D9542) staining on embryos fixed as above was for 1 hour at room temperature at 100ng/ml in PBS.

3. RESULTS

3.1. Survival of decapsulated embryos

Most stages of embryos tested in APW without supplements were unable to survive >48 hours; older embryos survived longer (if decapsulated >3 days post-fertilization). Some embryos survived for 24 hours only. All stages were compared to capsulated embryos from the same jelly mass. Average survival of multiple trials in APW is shown in Table 6.

<table>
<thead>
<tr>
<th>Stage</th>
<th>decap Alive after number of hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 cell</td>
<td>24  48  72</td>
</tr>
<tr>
<td>Gastrula</td>
<td>10%  0%  0%</td>
</tr>
<tr>
<td>Trophophore</td>
<td>80%  25%  0%</td>
</tr>
<tr>
<td>Veliger</td>
<td>44%  37.6%  20%</td>
</tr>
<tr>
<td>Hippo</td>
<td>100%  100%  100%</td>
</tr>
</tbody>
</table>

Ficoll was added to increase osmotic pressure, as we hypothesized that decapsulated embryos lacked pressure. Only one trial was conducted, as shown in Table 7.

<table>
<thead>
<tr>
<th>Stage</th>
<th>decap Alive after number of hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trophophore</td>
<td>24  48  72</td>
</tr>
<tr>
<td>Gastrula</td>
<td>0%  0%  0%</td>
</tr>
<tr>
<td>Veliger</td>
<td>25%  25%  0%</td>
</tr>
</tbody>
</table>

The first ACF failed to enhance survival of the decapsulated embryos when monitored over 24 hours, as shown in Table 8.

<table>
<thead>
<tr>
<th>Stage</th>
<th>decap Control Decap-APW Decap-ACF1</th>
</tr>
</thead>
<tbody>
<tr>
<td>20-cell</td>
<td>100%  0%  0%</td>
</tr>
<tr>
<td>Gastrula</td>
<td>100%  0%  0%</td>
</tr>
<tr>
<td>Trophophore</td>
<td>80%  0%  0%</td>
</tr>
<tr>
<td>Veliger</td>
<td>100%  0%  20%</td>
</tr>
</tbody>
</table>
Additional ACF formulations (ACF2-ACF4) had numerous different alterations in attempts to best enhance survival. Numerous trails are averaged in each and the results are shown in Figure 2A-I. For all graphs, number of days is shown on the X-axis.

Figure 2A(Gastrula), 2B(Trochophore), and 2C(Veliger). Enhanced survival of decapsulated embryos in ACF2.

Figure 3A(Gastrula), 3B(Trochophore), and 3C(Veliger). Enhanced survival of decapsulated embryos in ACF3.

Figure 4A(Gastrula), 4B(Trochophore), and 4C(Veliger). Enhanced survival of decapsulated embryos in ACF4.
3.1. Immunohistochemistry and DAPI Staining

Immunohistochemistry using antibodies directed against PKC or tubulin, and staining for DNA with DAPI were performed in order to explore the suitability of this organism for these developmental biological techniques. After the staining with a fluorophore was performed in all cases, the results were observed both under epifluorescence and light microscopy. Nonspecific background and the visual challenge presented by observing the expression pattern of a cytoplasmic protein in an embryo multiple cell layers thick were issues with the immunohistochemistry (Figure 5A-D). However, nuclear staining with DAPI gave a much clearer picture, especially in stages younger than two days post-fertilization (Figure 5E).

4. DISCUSSION

Raising decapsulated snail embryos in APW yields a low survival rate among most embryonic stages. This demonstrates a lack of proper nutrients or osmotic pressure provided by the APW. We hypothesized that the decapsulated embryos required more protein and ions to supplement APW. This led to the development of ACF in order to enhance survivability. The hypothesis supporting osmotic pressure as the main component lacking in APW led to the testing of Ficoll. Ficoll supplementation yielded a slight increase in survival, confirming that osmotic pressure is critical to the proper development of a decapsulated embryo. However, these results also illustrated that osmotic pressure is not the only component provided by the capsular fluid for optimal survival. With these results, we were able to move forward with various formulations of ACF to test components critical to complete development.

The ACF results were compelling. While no ACF tested allowed for full development from one-cell stage to juvenile snail as ideally sought, ACF2, 3, and in particular, 4 (see Figure 4) greatly enhanced the survival rate compared to APW. Greater survivability was observed through the comparison of rapid cilia-mediated spinning and normal progression to the next developmental stage. These results suggest that the ACF produced in the laboratory did contain many of the proper nutrients and approximate pressure required for normal development. In a majority of the experiments, however, the embryos were not observed alive after 48 (for experiments begun at the gastrula stage) to 72 hours (for veliger and trochophore). Thus, although ACF provided components to enhance survival, it did not supplant the capsule to support complete development.

During observations, indications of possible contamination were present. Initially
ACF was stored at 4°C to minimize bacterial growth. After suspected contamination clouded the fluid, ACF was sterile filtered. However, a persistent film was present at the surface of the fluid. Thus freshly made and filtered ACF was preferred for experiments. As these results reinforce, when developing within the capsule, the embryo is not only provided nutrients and appropriate pressure, but also protection from contamination. The absence of a sterile environment may be one of the contributing causes of the cell lysis, dissociation, and eventual death of the embryos.

The preliminary immunohistochemistry experiments gave positive indications for detection of molecules in these embryos using these techniques. However, the background fluorescence exhibited by antibody staining of both cytoplasmic proteins would best be minimized by advanced microscopy. Currently, a nuclear stain such as DAPI is much more readily detected under simple epifluorescence and allows for observation of the number and characteristics of nuclei. All these results have served to further our understanding of *L. palustris* and increase its usefulness as a model organism for developmental biology laboratories and experimentation.

**ACKNOWLEDGEMENT**

This work was supported by a grant from the NASA WV Space Grant Consortium and Shepherd University for STEM undergraduate research. The authors thank Dr. Colleen Nolan for her helpful thoughts and insights.

**REFERENCES**


What Is Firefox OS?
Jordan Cannin, Chad Vanorsdale, Jessica Novak

Department of Computer Sciences, Mathematics, and Engineering
Shepherd University
P.O. Box 3210
Shepherdstown, WV 25443
jcannin@gmail.com, ctvano92@gmail.com, jnovak01@rams.shepherd.edu

ABSTRACT
The web and mobile market are two ever growing worlds. We now see a mobile ecosystem where there is great separation between operating systems. Applications currently available cannot extend across multiple mobile operating systems. Firefox OS breaks this mold. This latest mobile operating system is entirely web-based. Applications for this operating system can be deployed across all platforms which support the web.

KEYWORDS: Firefox OS, Mobile, Mozilla, Web, Gecko, Gaia, Gonk

1. INTRODUCTION
Firefox OS is an open-source mobile operating system built by the Mozilla Corporation. It was built in order to advance the mobile web by giving web applications better access to Application Programming Interfaces (APIs), especially hardware access based API. These "Web APIs" allow for web apps to achieve a desirable user experience comparable to the experience of native apps on currently popular mobile operating systems such as Android and iOS. In this paper I look at the architecture of Firefox OS and then conclude with the implications of this new OS.

2. Gonk
Gonk is the base layer of Firefox OS and serves as the porting target of Mozilla's popular Gecko browser engine. Gonk uses a Linux kernel and functions like a small operating system in which the traditional browser layers above it can rely upon. The bootstrapping process it uses is similar to many Unix-Like operating systems. The main process is the traditional Init process, but another important process exists the B2G process. Alongside the Kernel processes is the Userspace Hardware Abstraction Layer that provides access to device hardware for the Gecko layer. Below the different aspects are analyzed in more detail:

2.1. Kernel
The Firefox OS kernel is similar to an upstream Linux kernel, while also incorporating a few changes made by Android to their Linux Kernel. As this is an open-source operating system the vendors who provide the operating system (OS) may also make customizations, further differentiating non-stock versions of Firefox OS. Running alongside the kernel is a Userspace Hardware Abstraction Layer which abstracts access to the different hardware devices and drivers managed by the kernel, and this HAL provides upward movement of hardware access to the runtime layer. Like many other *nix systems the Init process is the first to start.

2.2. Init
Firefox OS is currently using a modified version of the stock Android init.rc to
implement their init process. The init process differs from other systems in that it not also starts the key Firefox OS processes. One of these is the B2G process. This is the core process of Firefox OS and is required for it to function.

2.3. B2G

The B2G process, named after Firefox OS's original codename Boot-To-Gecko, is the primary process in Firefox OS and has high level privileges. This process is allowed to access most of the individual hardware devices found within the overall system and is responsible for web apps being able to access things like the telephony stack. This process is responsible for communicating with the modem, drawing to the display frame buffer, and all other low-level tasks relative to the overall operating system. It also implements the Gecko layer found above the Gonk layer.

2.4. Content Processes

The B2G process is tasked with spawning many low-rights content processes that handle specific tasks. This is how web applications are loaded as well, as low-rights content processes only able to access the most basic needs with most of their work abstracted out and secured by the operating system. The rild process is spawned in order to interface with the modem, drawing to the display frame buffer, and all other low-level tasks relative to the overall operating system. It also implements the Gecko layer found above the Gonk layer.

3. Gecko

Gecko, is the runtime environment that handles the processing of the web code. It contains the virtual machine that is used to interpret and run the Javascript code that is passed to it from the applications. It also contains the rendering and display engine for the CSS and HTML code. Gecko provides for all the necessary translations of code to allow for it to be passed directly to the hardware and run with less middleware than a traditional browser [2].

The b2g process, which is launched by the init process, is a center, primary piece of the system. It is able to access most hardware devices and features, and runs the Gecko layer itself. B2G is currently a single process system. One Gecko engine responds to the UI, Gaia. The Gecko engine also shares resources in order to load web applications and web pages simultaneously. Development is currently underway to implement a multiprocessor Gecko system. This would break the Gecko engine into several partitions. A single master Gecko process would run and respond to Gaia. Several separate slave content processes running applications and web pages would exist alongside the master process. These processes would communicate through the implementation of IPDL. IPDL is currently used for processes in Gecko to communication, and it is able to pass messages across multiple threads [4]. Gecko also provides the necessary components for layout and graphic processing so that the Gaia
layer can be developed using CSS and HTML to display its user interface.

4. Gaia

The third layer is Gaia, which is the user interface written in CSS, HTML, and JavaScript. Everything drawn to the screen is part of Gaia which then passes this code to Gecko to be interpreted and rendered. Gaia allows for the display of the applications and for the tweaking of the user experience. It is through Gaia that the user will see what they can interact with in the operating system, and hacking Gaia is how you can change the interaction of users with the underlying operating system. Gaia is designed to be open-source and platform independent so long as the platform you are porting it to supports the API’s it uses. All the API’s used by Gaia are currently open API’s that are available to any other web engine to implement making a mobile operating system based on other web engines such as webkit plausible [2].

5. Web API’s

Due to the fact that Firefox OS is based off of web languages, web API’s are central to this operating system. The Mozilla WebAPI team has an extensive list of web API’s planned for the initial release of Firefox OS. These API’s include your basic phone functionality such as calling, messaging, phone settings, device storage, battery status, alarm, clock, Wi-fi information, mobile connection/signal strength, contacts, web browser, vibration and bluetooth capabilities. The mouse lock API, included in basic phone functionality, allows for basic touch screen capabilities. The cursor is hidden, touch is recognized, and location and positioning of the finger become the cursor. The power management API, another basic mobile phone feature, allows the device to recognize certain stages of use. These stages include the powering up and down of the device, idle, lock screen, screen unlocked but applications left idle, and so on. Power management also enables the feature of backlight control.

Other web API’s, also planned for the initial release, bring the typical features of a smartphone to the operating system. Idle and push notifications APIs are two examples. The idle API allows the user to receive notifications when the device is left idle, and the push notifications API allows notification messages to be sent to various applications. Screen orientation, network statistics/data usage, network information/connection speeds, web payment, application permissions, FM radio capabilities, and geolocation/GPS are also included smartphone feature web API’s. Ambient light API is a typical feature of most smartphones today. The API can recognize the lighting of the surrounding environment. If the environment were to be well lit the ambient light API can sense this and respond accordingly. The backlight can respond and can dim, helping to save battery life of the device. The last to mention of the common smartphone functions is the proximity sensor API. This sensor enables the device to recognize the distance of objects. For example, if the user were to lift the device to his or her face to place a call, the proximity sensor would recognize the location of the nearby object, the face, to the screen and lock itself. This allows for increased battery life and reduces in accidental touching of the screen.

There are several API’s to note which offer a richer user experience. The open webapps API may not be unique, but it will offer an increase in user experience. This API allows for the download of open web applications from the application store. These applications
will be downloaded and stored client-side on the user’s phone. This API will also allow the user to manage any downloaded applications as well. Next there are several API’s which deal with file management. These API’s create a richer user experience by increasing phone functionality. First is the archive API which enables the ability to read zip files which may be encountered on the web. The file handle API allows the system the ability to write to a file. This API is supported in the IndexedDB API. IndexedDB allows for data objects to be stored locally. This is incredibly important for an operating system which uses the web as a platform. If a network connection is to fail IndexedDB API can provide for basic phone functions to continue operating without interruption.

There are also several API’s already planned for the future, after the initial release, and are currently being developed. Most notable is the WebRTC effort which encompasses capabilities spread across two APIs, camera and peer to peer. The functions which would be included are as follows: the secure transmission of a video call within a web browser, the ability for the user to place a video call from inside a social API, calling a user from inside an application and sharing images, files, etc with that user through a P2P connection, and allowing for video, audio and data sharing between users within an application. Other future API’s are planned to improve functionality at the high-level and low-level. High-level API’s include: calendar API, spellcheck API, resource lock/prevent resource turnoff API. Also included is the webNFC API. As its name implies it will allow for NFC technology on the device. More high-level APIs include: Keyboard/IME API, which simply enables the use of a virtual keyboard, LogAPI and background services API. Both APIs deal with user activity. LogAPI would have the ability to track and log user activity on the device, while background services allows for activity in web applications to continue to run in the background. Last to note are the low-level APIs: http-cache API, webUSB API and udp datagram socket API. The http-cache API would be the tool necessary for the user to search what is stored in the browser, add/remove entries from this list, and obtain data directly from cache. Next is the webUSB API which allows for the USB device driver to be implemented with javascript alone. Lastly is the UDP datagram API. This API enables the UDP protocol to be used for the exchanging of messages over a network.

When planning for the development of these web API’s several design principles are taken into account. The first design principle considered is security. API’s must have sufficient security built into them before they can be deployed. For example, it could become detrimental if a web application were able to gain access to personal user information, such as, contacts, location, etc. Confidential information must be thought of and taken into account, along with potential threats and their risk levels. Another design principle is the choice of low-level APIs versus high-level APIs. Low-level APIs focus on how to provide support for hardware components and how to implement certain technologies, like the USB device driver. Priority is placed with the development of low-level APIs. High-level APIs, on the other hand, provide different types of functionalities, like camera access [3].

REFERENCES


Firefox OS
Web App Development
Chad Vanorsdale

Department of Computer Sciences, Mathematics, and Engineering
Shepherd University
P.O. Box 3210
Shepherdstown, WV 25443
CTVano92@gmail.com

ABSTRACT

Native apps continue to be allowed better access to device APIs while the mobile web is denied this privilege. This is solved partly by hybrid apps that combine the capabilities of web apps and native apps, however performance and usability can become an issue. Due to this Mozilla has created Firefox OS which aims to give web apps the same access as native apps and create a native experience using only web technologies. This paper analyzes Firefox OS and how it solves problems currently surrounding mobile web apps.

KEYWORDS: Web App, HTML5, Firefox OS, Web API, Mozilla

1. INTRODUCTION

iOS and Android, the two leading mobile operating systems, continue to lock web applications out of their "App Stores" and prevent them from gaining access to device hardware. Native Applications are exclusively granted these privileges giving them an unfair advantage in discoverability, performance, and usability. Even as developers are beginning to turn to HTML5 to provide for a unified codebase, they must wrap their web apps in native code in order to gain the same privileges, causing an unnecessary extra layer of code which degrades the web apps performance.

Firefox OS by Mozilla is eroding these barriers by making web apps their platform's de-facto standard. The entire platform runs as a browser and web apps have full access to hardware allowing them to perform much like native apps. This paper looks at developing web apps for Firefox OS while comparing it to iOS and Android.

2. PROGRAMMING LANGUAGES

The programming languages used by Firefox OS are HTML, CSS, and JavaScript. These are your typical languages to create a client-side application stack for web applications. HTML is used for structure, and Firefox OS is relying on the new HTML5 standards which provide for rich new elements that expand HTML and reduce the need for plugins. Without a reliance on plugins, the web is freed from compounding extra layers of code that do not naturally flow with the rest of your content. Examples of new options provided by HTML are the video tag for native video, the audio tag for native audio, and the canvas tag for native graphics. Once you have programmed the structure of the page with HTML, you may style it with CSS.

CSS3 is the primary standard for web apps currently and provides for new, more powerful ways to build apps. CSS2 allows you to create layouts and design your web app by specifying colors, fonts, margins, etc. With CSS3 you can expand upon this and use...
multiple background images for more interesting interfaces. You may also use native gradients so that you have less graphics to import. Basics have also been reworked by adding simple but useful options such as rounded rectangles, which are used throughout Firefox OS to create a circular app icon on the launch screen. Once you have styled your app you then can add functionality to it with JavaScript.

JavaScript is a dynamic, JIT language that easily interfaces with your HTML and CSS. You can manipulate the current layout through nodes in a structure known as the Document Object Model (DOM) or provide controls for more dynamic elements such as your audio, canvas, and video tags. With Firefox OS many new JavaScript APIs have been introduced to enhance the usability of the web for web apps as well as to improve the user experience by proving for APIs that connect your web apps to the device hardware.

3. NEW WEB FEATURES

Firefox OS is tasked with making web apps more accessible to the overall system which the web ecosystem doesn't currently support. Firefox OS solves this problem by supporting new Web APIs as well as Web Activities. Both of these are accessible to web apps created for Firefox OS and allow for your web apps to have the same feelings and interactions as native apps on other platforms.

Open Web API's are abstract interfaces available to JavaScript that give access to hardware. There are many new web API's available such as for vibration, battery status, alarms, and bluetooth. While these are only some of the ones available, they are an example of features that your web apps can access that would not normally be accessible to web apps on other platforms. This means that your web apps are unrestricted and have a full user experience.

Web activities are similar in that they allow your web apps to remove themselves from isolation and interact across the system. You can design your web app to interact with other apps to complete tasks such as providing pictures or text to the calling app. The calling web app can also specify specific activities it would like to do such as pick an image or bring up a default app for a certain type that is linked. Available activities include:

- configure
- costControl/balance
- costcontrol/data usage
- costcontrol/telephony
- dial
- new
- open
- pick
- record
- save-bookmark
- share
- test
- view

4. MOZILLA MARKETPLACE

Mozilla has started to release access to their new Mozilla Marketplace which is a central location to install and find web apps. This marketplace will be available as the primary app store in Firefox OS. Mozilla will also be allowing third party app stores, so a company such as Google may decide to and be able to have their own Chrome Web Store available on Firefox OS as a secondary web app store.

The Mozilla Marketplace will be available not only on Firefox OS but also on other devices running Firefox such as Android. The marketplace will feature two primary payment models on launch which are upfront purchases from the marketplace and then in-app purchases once you download an app.
The marketplace will function similar to current app stores with apps being delineated by certain categories, filterable by specifications, and searchable by parameters. Users will be able to rate web apps so that web apps have the option to stand out from the crowd and developers may gain popularity for developing well-built apps using web technologies. When submitting your app to the Mozilla Marketplace, a hybrid admission approach will be used that combines the approval process of Android and iOS. A system will first check your app to make sure no critical permissions are attempting to be used, and then if it detects that you are attempting to use critical permissions, such as accessing the ability to shutdown the device then it will be moved to a queue to be reviewed by a human review. The marketplace will provide for an easy way for web apps to be found, delivered, and rated so that users have an easier time finding and using them and so that developers can get their apps to users without hassle as well as receive more critical feedback.

5. COMPARISONS

In order to better understand current web app performance three tests were used to test web apps on Firefox OS, iOS, and Android. An iPhone 4 running iOS 6 was used to test web performance for iOS and a Samsung Galaxy S2 running Android 4.0.3 was used to test the performance of the web on Android. The Samsung Galaxy S2 was wiped of the Android ROM and replaced by Firefox OS to further test the results of Firefox OS. Each test was run three times and the average score was charted and compared.

The first test used an online tool called Peacekeeper to compare cross platform browser performance. Peacekeeper is a free online benchmarking tool from Futuremark that measures JavaScript operations per second and recorded frames per second. A higher score measures better overall performance and the score is based on an aggregation of test results performed. The Peacekeeper test revealed the difference between native and hybrid apps. On iOS, the Chrome browser is a hybrid app while Safari is a native application. The inherent performance degradation found within hybrid apps is the primary factor in the score difference between the two browsers. Also of the note, the test revealed that the differing hardware of the iPhone 4 and Samsung Galaxy S2 caused largely different results between the platforms, therefore it is best to study the results of Android versus FirefoxOS for direct comparisons of performance. FirefoxOS was unable to outperform the browsers on Android which leads to the conclusion that it needs further optimization, which may be found when official phones are released.

The second test was an online test that simulated Asteroids. It tests typical 2D rendering performance for commonly used operations. After performing a series of graphical renderings, an aggregate score is given to measure performance for different platforms. Firefox OS again can not outperform the current Android browsers, including Firefox for Android which further
shows a lack of overall optimization which could be explained due to its pre-release stage and lack of official phone.

The third test was a simulator of a web based game running called Impact. This tests realistic performance of web apps. The test simulates running a game and measures any noticeable pauses due to actions such as garbage collection and rendering. The test also measures frames-per-second (FPS) to determine and overall score for performance. Firefox OS yet again proved to be unable to top other competing web environments on Android.

Firefox OS does not outperform the other web environments on the charts, but does maintain a moderate performance. A major problem is that Firefox OS was put on a Galaxy S2 which was not optimized or meant to run Firefox OS which could affect performance as well as Firefox OS still being in rapid development.

As seen on the Impact test, the Firefox browser running on the same phone with Android as the OS has the greatest performance. This means that Firefox had the best performance in realistic situations and Firefox OS could potentially have better performance on its own device if optimized similarly to Android.

6. CONCLUSION
Firefox OS is a solid operating system that is stabilizing as the operating system continues to be developed. It provides more features for the web and improves the web apps across its system. Although Firefox OS may not be the best performing mobile operating system, the Web APIs they are working on and the model for a web based mobile OS are the most important part. If all current operating system work on implementing these into their operating system, it will be a better future for web apps on all systems not just one.

REFERENCES


ABSTRACT
Firefox OS has coined itself as a new mobile operating system with making its platform the open web. This approach is unlike any other mobile operating system. The concept of making the open web a platform is new. Our research aims to investigate the security of this new approach. We have examined specific areas of the mobile operating system. Through this examination we suggested security improvements to be deployed on Firefox OS before its official release.

KEYWORDS: mobile, security, encryption, file systems, applications, permissions, operating system

1. INTRODUCTION
Firefox OS is a new mobile platform developed by Mozilla. The operating system is still in beta and has yet to be officially launched. Firefox OS is entirely developed around the standards of the web. It aims to be a completely open platform which can run any web applications. Our research posed the question: how safe is an operating system which does this, and could security improvements be made to the operating system as a whole before its official release? The goal of our research was to exam Firefox OS against other popular mobile platforms. We choose to do this comparison against Android and iOS because they have the vast majority of the market share. Mobile devices are more commonly used today than ever before. Mobile devices typically hold more personal information than what a home computer now does. It is important to understand the overall operations of mobile operating systems for this reason. Our research examines how this personal information is stored and handled. We explored each level of security on the three mobile operating systems. We partition our research into three areas which included architecture, encryption and file system standards and application handling.

2. Methodology
We first wanted a hands-on experience with the Firefox OS operating system. The source code for the operating system is online and available to be downloaded. We took advantage of this and downloaded the source code ourselves. We used a Samsung Galaxy S II for the test of placing Firefox OS on an actual mobile device. We flashed the phone and removed the Android operating system. Once the phone was equipped with Firefox OS we first tested the overall functionality of the device. We examined questions such as: how easy is the device to navigate, how well do specific applications perform, how secure does this device appear to be, are any security functionalities visible, and so on. After having a hands-on experience with testing the device
we broke our research into categories. We examined the integral parts of a mobile operating system when it comes to security: architecture, encryption and file system standards and application handling. We looked at each category for each mobile platform: Android, iOS and Firefox OS. We compiled all the information acquired into charts as a simpler means of comparison. After comparison of the information we obtained we gave an overall evaluation of Firefox OS. For the final phase of our research we gave a set of security improvements that we suggest Firefox OS make before the official release of the operating system.

3. Results
The following are our results. Each category processes its own section of detailed information. Below we state and explain our findings. We have included our charts, and provided a detailed explanation to accompany them. The following section includes all the research we obtained throughout this project and process.

3.1 Mobile Security Model
Each mobile platform possesses its own unique, security model. These security models show the internals of each mobile operating system. The models represent to us the structure of each device. They also show us the internal flow of data. Below we show visual representations of each model to give a clear picture before we being a more detailed explanation of each operating system.

Figure 1. Android Security Model:

Figure 2. iOS Security Model:
3.2 Device Architecture
In this section we examine the architecture of each operating system. We will explain how the hardware and structure impacts the security of each operating system.

3.2.1 Android Architecture
Android is built on a Linux kernel. Key UNIX features have been implemented to create a safer operating system. The key security features of the Linux kernel are as follows: process isolation, user-based permissions and secure IPC. The Linux kernel offers secure communication between applications which run in different processes. The kernel also ensures applications will not harm one another, which prevents harm to personal data [1].

As the security model shows, the system partitions the Linux kernel, libraries, application framework and application runtime. The partition is read only. It cannot be modified by an external source; therefore malicious code cannot harm the key components of the system [1].

3.2.2 iOS Architecture
Apple created, what is called, a secure boot chain. This secure boot is a method for ensuring at each step of the boot-up process each component is verified as secure and as an Apple component. The bootloaders, kernel, kernel extensions and baseband firmware are all signed by Apple. Each component contains an Apple Root CA public key. This public key is given to each component to verify software at the lowest levels have not been altered. The iOS operating system can only run on Apple devices, and in turn, Apple devices can only run iOS. If in the secure boot chain process one step cannot be verified the operating system will halt and enter recovery mode. This method is used to ensure the integrity of the operating system. It makes it impossible for older versions of the operating system to be loaded onto the device to exploit old security holes [8].

3.2.3 Firefox OS
Firefox OS, like Android, is built on a Linux kernel. At the lowest level exists the kernel along with OEM drives and OEM Modem Firmware. The init process, commonly heard of with Linux, mounts the file system and begins system processes. It is the process manager. The operating system is divided into three layers: Gonk at the lowest level, Gecko and Gaia as the user interface. The operating system employs two processes called core and content. The core process is in charge of controlling the access web applications are granted to resources. The content process is the space in which those web applications run. The core and content processes communicate
through an encrypted channel. The runtime security model is an important piece of this operating system. Its jobs include enforcing application permissions, protecting the core and content processes and ensuring the integrity of the communication channels [10].

3.3 Encryption & File Systems
The file system is core to every device. It contains all files and data associated with those files. Security comes to the file system through the use of encryption standards. File systems also have predefined permissions set in place for protection as well. Each operating system has a different approach to implementing encryption and file system security.

3.3.1 Android Encryption & File System
Android has created a set of cryptographic APIs to be used by applications. The encryption methods used are as follows: AES, RSA, DSA and SHA. The file system in its entirety is encrypted. All user data is encrypted within the kernel. The file system is encrypted with AES 128 with CBC and ESSIV. A SHA 256 encryption key is encrypted by AES 128 as well. The key is derived from the user’s device password. This mechanism was set in place as a way to prevent unauthorized access to device data. The device password is also protected against brute force through repeated hashing with SHA 1 using the PBKDF2 algorithm. This process takes place on the device password before the file system key is encrypted. The file system also has its own list of defined permissions to accompany its encryption. Included within the permissions is, files that are accessed and read by one application cannot be intercepted and read and/or modified by another application. Each application runs within its own process or space. These permissions were set in place to ensure one user, who may have created an application, cannot alter the files of another user, who may have downloaded the application [1].

3.3.2 iOS Encryption & File System
In the operating system there is a built in AES 256 crypto engine. This engine exists between flash storage and the main memory system. SHA 1 is also used and is implemented within the hardware of the device. AES 256 bit keys are placed within the application processor, burned into silicon. This method makes these keys unattainable. They are kept from both device software and firmware. The keys are kept safe from outside interception and are only readable by the crypto engine. Any other key found within the system is created with a random number generator. A mechanism called Keychain Data Protection is found within the system as well. It is a SQLite database which can be found in the file system. It handles all sensitive data. This data includes keys, passwords and login tokens. Keybags are also implemented. Keys are divided and placed within four different categories which are: system keybag, backup keybag, escrow keybag and iCloud backup key bag. Files are partitioned and encrypted with a pre-file key. The pre-file key is 256 bits and is handed off to the crypto engine. The engine encrypts the file using AES and CBC. The pre-file key is then wrapped using AES key wrapping. All pieces are stored together and placed into the file’s metadata. The entire package is then encrypted for a final time with the file system key [8].

3.3.3 Firefox OS Encryption & File System
Firefox OS has no encryption standards set in place for its file system. There are, however, file system permissions set into place. These
permissions include an ACL (Access Control List) and DAC. Both are derived and based off of Linux developments. The ACL enforces file system privileges. DAC is the file system permission model which stores and determines what system privileges are accessible. If access is granted to the file system Web APIs are used to obtain data. APIs such as IndexedDB API act as storage systems as well. There data is obtained from the file system and stored outside of it for easier accessibility. The data can be assumed to be low-privileged, insensitive data [11].

3.4 Application Handling
The core of all mobile devices is downloadable applications. Available applications bring personalization to each mobile device. These applications often contain personal information and sensitive data. This can include location-based information, credit card information, banking information, work and home addresses, etc. Applications themselves must have their own security features to protect end-users data.

3.4.1 Android Application Handling
The application sandbox is the main way in which Android handles application security. The sandbox exists at the kernel level. Each application runs within its own individual process. Multiple applications cannot run on the same user permissions. The kernel is responsible for the enforcement of security between applications. To keep track of applications each is assigned a unique user ID (UUID). These IDs are assigned by the kernel and tracked by the kernel. The application sandbox is based around the UNIX style “user separation of processes and file permissions” [1]. All applications operate within the sandbox. If memory corruption were to occur within an application it will not affect anything outside of its self, due to the use of the sandbox. Applications are, by default, unable to communicate with one another. One application is unable to access information belonging to another application. Information can be exchanged between applications only through user permission. All software existing above the kernel, like system libraries, run in the sandbox as well. Every application is required to be signed by its developer. This signature acts as an agreement with Google. If any harm comes from an application, the developer will be tracked by Google and held accountable [1].

3.4.2 iOS Application Handling
Runtime Process Security is set in place for applications. Each application is given its own home directory for its files. Third party applications are sandboxed. This can be bypassed through the use of APIs provided by Apple. These APIs give applications access to one another, only through mutual application permission. The entire operating system mainly executes in non-privileged user “mobile” mode. The entire OS is mounted as read only. This means system files and other resources are kept from user applications. They cannot be accessed. All applications must receive Apple issued certificates. Applications from websites cannot be downloaded onto the device without an Apple signed certificate [2].

3.4.3 Firefox OS Application Handling
All applications running on the operating system are titled as open web applications. The trademark of the operating system is to run any application written in web languages. All web applications run in a content process, as previously mention. The content process is given low privileges. Firefox OS uses the sandboxing method. The sandbox receives requests for permissions from the application; it then forwards those requests to the “Gecko server” process. The “Gecko server” will
make the decision on granting special permissions to the application. The reply from the “Gecko server” will travel back through the sandbox. This method protects against other applications trying to intercept messages between the application and server to gain those requested special permissions [11].

4. Research-Based Recommendations
After researching each platform from the system level to the application level we have been able to effectively compare and contrast each mobile operating system. Our research into mobile malware also helped us to determine which application handling practices work and which do not. From the above research we were able to compile two areas of security enhancements for Firefox OS. We believe that these enhancements should be put into place before the official release of Firefox OS.

4.1 Security Enhancements
Upon researching both the Android and iOS platforms we found that both had encrypted file systems. Firefox OS, however, does not have any encryption standards set into place. We suggest encrypting the file system with Crypto APIs, as seen in Android. We believe this would be the easiest method of implementation. The Crypto APIs are able to deliver many methods of encryption including AES, RSA and SHA. We suggest basing the file system key off of the device’s user password after it has been repeatedly hashed with SHA 1. Repeated hashing will protect the password, and the overall file system key, from a brute force attack. We also suggest a storage system for encryption keys and certificates. Currently Firefox OS uses an ACL (Access Control List) and DAC for file system access. Firefox OS should continue to implement these two methods of access protection.

Developers are currently coding a Bluetooth API for the operating system. This API is being created using Bluez, a Linux Bluetooth protocol stack. Neither the API nor the protocol stack has any encryption standards in place. We believe an encrypted communication channel is an important feature for the operating system. Encryption Mode 3 is currently the most secure standard for Bluetooth transmissions. Mode 3 uses link level enforced security. Link level security first uses a PIN code that is usually selected by the user. After PIN codes are entered by both devices wishing to connect a link key is established and used for authentication. Lastly, an encryption key is used between the devices to enable encrypted transmission of data.

Lastly, there is room for an increase in networking capabilities. Firefox currently has no VPN capabilities. There are no APIs in development or planned for the future. VPNs provide long distance, secure network connections. VPNs are commonly used by businesses so end-users can connect to a secure business network. Smart phones have gained wide use and acceptance from everyday users. The next frontier for the mobile world is the business and enterprise level. Enabling encryption standards and such features as VPN technology allows Firefox OS to be a competitor in the corporate, mobile world.

4.2 Application Security Enhancements
As we conducted our research we found that Firefox OS lacked security in their application layer. After we looked at the three different operating systems we put together suggestions that Mozilla should look into and possibly incorporate into their final product.

Firefox OS uses JavaScript, CSS, and HTML for their application architecture. These web languages have been used for over 15 years
making them very popular for web development and will be used for the foundation of Firefox OS. However; that doesn’t mean these languages have become more secure with age. As JavaScript runs on Firefox OS, it will allow the device to be potentially exploited by hackers and/or attackers since new programs are constantly being written to utilize the vulnerabilities of this language. Another disadvantage to using JavaScript is that it can be interpreted differently on different browsers. Even though Firefox will install its own browser as a native app, you will be able to install any other browser available in the app store. If the code was not written (by the individual app developer) to be cross compatible it could leave security vulnerabilities making the application simple for someone to access secure data on your device.

Another issue which presented itself was Firefox OS’s permission model. They use a set of standards they call Principle of Least Permissions. The model states that they will “start with the absolute minimum permissions, then selectively grant additional privileges only when required and reasonable” [11]. We feel that this could be an issue. In Mozilla’s description of the permissions model, they never state what the “minimum permissions” are or what “required and reasonable” will entail. Even if they did define these restrictions more clearly, we believe that the user should be in control of any and all permission granted.

Lastly, we would like to see a security feature of iOS implemented into Firefox OS. iOS has incorporated a security feature called Address Space Layout Randomization or ASLR [2]. ASLR works by randomly arranging the positions of the data on the RAM. This technique lessens the chance of an attacker being able to find the location of the specific information they are looking for. By increasing the space and the fact that it is randomly allocated to that space, it is ultimately keeping your data more secure.

9. Conclusion

We found that Firefox OS does have useful security features. Creating an operating system completely developed around web standards has not hindered the overall security of the operating system. There are areas which we discovered in our research that need further work and more improvement. These areas include the incorporation of encryption standards and improved application handling. Through these improvements we believe the operating system will be capable of competing with the Android platform, and be capable of competing in a world market.

REFERENCES


Advanced Gesture Identification:  
IGIS Interpolated Gesture Identification System  

Randy McCright  

Department of Computer Sciences, Mathematics, and Engineering  
Shepherd University  
P.O. Box 3210  
Shepherdstown, WV 25443  
rmmcct01@rams.shepherd.edu

ABSTRACT  
Gesture Identification is a robots or other computer systems ability to detect and purposely track and detect ordinary user input such as physical movement as well as sound production. These inputs can be used to control various computer systems. With the development of sensors, improvement of computer abilities, and creation of recognition algorithms, gesture identification can be utilized in more everyday situations. The capabilities of gesture identification will be evaluated more explicitly through research, and then implemented in a useful manner for individuals whom are handicapped. Specifically those with limited mobility. Application examples are deep-seated in environmental manipulations within a person's home such as temperature, lighting, and sound control. More advanced solutions can control intricate systems such as laundry and kitchen appliances as well as personal computers.

KEYWORDS: Gesture Identification, skeleton tracking, depth evaluating algorithms, Linear Interpolation

1. INTRODUCTION  
As computer and robotic systems progress there will become a greater need for more advanced methods of interaction between humans and these embedded systems. Having quicker communication capabilities will improve human safety in many industrial applications as well as system efficiency when completing time sensitive tasks in both industry and consumer applications. Gesture tracking, attempts to distinguish all forms of physical user input. For example hand motions, facial changes and body movements as well as sound stimulus can be tracked and logged by the computer. This data can be transformed from low level 1's and 0's into significantly more understandable representations such as video streams and visual graphs to increase the interaction quality between the human and computer. These communication methods could potentially provide faster and more useful data communication between end user's and the computers or robots in question.

There are several types of sensors used in gesture identification but the three main types are depth sensors, RGB cameras, and microphone arrays. Depth sensors, sometimes referred to as time of flight sensors, can ascertain the distance of objects seen within the viewing angle by using an infrared(IR) laser.[1] The laser can determine distance by tracking the time it takes for reflected IR waves to return to the sensor. The RGB camera basically gathers color video to be used in grouping with depth maps to create important data for gesture tracking. RGB camera data and depth maps are filtered using specific techniques to separate objects from one another, such as a person and a table. With the added dimension of audio tracking voice
recognition can be used to determine the difference between one person and another. Using microphones in arrays containing multiple units facing in different directions the location of sound can be determined and applied to the tracking.[2,3,4] Other forms of sensing will be utilized in this research to determine differences in users as well.

The main user detected by one of these cameras would take control of the system: this means that he is the only person that can command of the program using gestures. Multiple users may take control at the same time as well if so desired. There are also many forms of feedback available from the gesture identification system such as a television system.

Several algorithms are required to make identification successful. For instance, the Expectation Maximization (EM) algorithm is used to fill in gaps in received data.[2] If for some reason the gesture identification temporarily loses track of a feature the EM algorithm will help to maintain accuracy until identification continues. Other methods include the Gaussian mixture model which is used to separate the background from an individual by comparing static data such as a wall from changing data such as a moving person.[2]

A major part of this study will be dedicated to the classification and understanding of the algorithms that make identification possible. Once these steps are complete it will be possible to manipulate these methods or add other layers of programming to achieve specific results based off of natural user input. The end goal is to provide a high level gesture input solution to enable handicap persons to more easily complete standard daily tasks. Other research in the area of gesture identification uses similar methods but also runs into common problems that this research will attempt to solve.

2.PROBLEM IDENTIFICATION

Even though gesture identification has come a long way in the past ten years there are many features that need to be improved. In some cases we need to create new features from scratch to make this technology suitable for consumer use's other than entertainment applications.

There are limited tracking abilities and a small number of specifically recognizable gestures. The two common gestures are based in the areas of waving and clicking using hand motions. Often if you become too close to the sensors depth tracking is lost. The depth camera also has a smaller width and height resolution than the RGB camera. This creates issues with tracking around the visible corners.

Most implementations of identification have a very limited number of gestures that can be recognized and this is the main problem that needs to be overcome. In a consumer environment there needs to be many available gestures so the number of controllable devices becomes larger. Adding features to gesture identification is proposed to make this technology useful in day to day life. Before gestures are added the algorithms that make gesture identification and tracking possible need to be improved to maintain specific and reliable reading of a users intended input.

3.OUR APPROACH: Interpolated Gesture Identification Engine IGIE.

OpenNi is a software dedicated to creating natural interaction between human and machine. The framework is designed to work with the C, C++, and Java programming languages. This open source software will be the basis for all algorithm improvement and addition thought this
research. Examples of the software's basic gesture identification and tracking can be found below in Figure 2. Three examples of RGB depth camera sensors are also shown. The Microsoft kinect sensor (in the middle) will be utilized throughout the research for its added sound detection capabilities. Later on in the research we switched from the kinect to an Xtion Pro this can also be seen in the picture below. (bottom)

![Figure 2. Gesture Tracking [5]](image)

OpenNi framework is the link between the sensors which generate raw data and the middleware software that exploit this data. OpenNi also bridges the middleware and application layers together. Middleware algorithms give added function to the raw data by implementing identification algorithms; for example to track a person's hand. Later on we demonstrate our use of the low level algorithms in combination with our own middleware for gesture identification. The utility of the final application is directly related to the methods by which OpenNi manipulates raw data and by how the middleware uses the received altered data to implement gesture identification and tracking. The necessary areas of improvement are both the OpenNi framework and Middleware software. Figure 3 illustrates a visual outline of the software and hardware levels of the identification and tracking system.

3.1. PROCEDURE

Because of limitations with the OpenNi framework the first step of development and research will entail advancing low level data algorithms to increase the versatility of gesture creation using middleware. Skeleton tracking for instance can detect the position and orientation of a person's limbs, head and torso but has several limitations when a person is turned sideways or backwards. Manufacturer's own research shows a maximum use of up to 45 degrees. While the skeleton tracker is implemented using middleware the root of the problem is based in the lower OpenNi data acquisition layer. Not only does the framework have trouble synchronizing live video feed data and depth sensor data but has issues with continually tracking individual users. As a consequence final applications are limited to a small number of gestures and their flexibility diminishes. This will be another aspect of gesture identification we will discuss more in this paper.

The first step of our development is to carefully examine the algorithms of the OpenNi low level layer and the middleware layer to determine how much improvement is necessary. In some cases it is expected to require development of entirely new algorithms. One of the very first additions will be to add logical panning capabilities to our sensors. This will be done to overcome issues in tracking when a user is positioned at the edge of the depth sensors field of view. We will achieve this goal by manipulating the OpenNi layer to not only keep track of the data that represents the individual but specifically their exact midpoint location relative to the field of view. The middleware layer will be able to
utilize this location to move a servo that will be attached to the sensors base; and thus keep the user in the most accurate viewpoint.

Our goals will include but not be limited to improving the low level OpenNI framework and middleware software to detect individual fingers, head movement, 360° body rotation, clapping, and the raising of a hand. After both low level and middleware software's are improved upon a final application will be implemented to present to newfound capabilities of the gesture identification software. Below in Figure 4. you can see a formal flow chart outlining our development process.

3.1 MULTI ANGLE IDENTIFICATION

In the past determining a gesture of a person from any angle has been unreliable. The biggest reason being the type of algorithms used in gesture identification. The Hidden Markov model uses states that are comprised of a large amount of data. Our approach offers a simpler solution using linear interpolation. We are able to achieve a high detection rate of up to 90 degrees rotation from the detection sensor. This is done by performing a coordinate rotation around the origin. The Figures below show the equations used in this process.

\[ x' = x \cos \theta - y \sin \theta \]
\[ y' = x \sin \theta + y \cos \theta \]

FIGURE. Rotation Equations.

First we send all coordinate data back to the origin because the rotation is impossible without doing so. We use the angle that is made from the users shoulder in relation to the sensor origin to determine an amount of rotation. After the rotation we send the points back to the original location only rotated. Doing this allows all data being sent to the Interpolation Gesture Identification System to be simulated as if the user is not angled away from the sensor at all.

3.2 USER PROGRAMMABLE GESTURES

In the past gestures were not customizable and were confined to pre-programmed gestures because of the way they were being recognized. Our method of linear interpolation allows gestures to be created on the fly and be as small to store as a few kilobytes. As mentioned before each gesture is determined by interpolating the current location of a users hand and comparing it with the location that the users hand needs to be. When gathering the data for a new gesture much auto leveling must be applied. This is done so that the interpolation process is more predictable. For instance the gesture corresponding to a circle would be more of an octagon according to linear interpolation.

First the user activates the detection process. He or she will hold their hand(s) still and allow for the software to guarantee their hand position. Next the user will have to move the hand(s) in the motions that they desire to set as the linear interpolated movement. Finally the user needs to hold their hand still as to signify the end of a gesture creation.

4. NEXT STEPS

The next part of the research and development process will be to determine the maximum number of users that can be detected at a given time under the new software. This will allow us to develop a strategy to make this technology usable on a larger scale. Also other aspects such as being able to determine if a person has left and re-entered. This is an issue because users who re-enter are often considered a new user.
REFERENCES


Extending Independence: 
RFID Tracking and Object Avoidance Robotics to Aid Persons of Disability
Michael Skaggs and Dr. Seung-yun Kim

Department of Computer Sciences, Mathematics, and Engineering
Shepherd University
P.O. Box 3210
Shepherdstown, WV 25443
mskagg01@shepherd.edu

ABSTRACT

The purpose of this project is to develop robotic aids for persons of disability. These robotic aids will be designed to take commands from a user and complete assigned tasks while avoiding obstacles that may stand in their path. An example of this could be as simple as a robot fetching a remote for an elderly member who may have impaired motor skills. The interface that we propose must be human-centric so it can be picked up and easily understood. We will also develop and implement a way to keep the robotic aid charged at all times, which will be explored as the project moves forward. The robotic aids will be “hexapods” to eliminate any problems that may occur, should up to two legs fail. Eventually, we hope these aids will have the ability to climb stairs, navigate walkways, and numerous other advanced object avoidance strategies. This will be a leap into elongating the time that people with disabilities can live independently.

KEYWORDS: ROBOTICS, RFID, OBJECT AVOIDANCE, DISABILITY, AUTONOMOUS NAVIGATION

1. INTRODUCTION

Radio Frequency Identification, or RFID, is a wireless identification system that uses electromagnetic fields to transfer data from a tag attached to an object. This is the base system that will help our robotic aid track down objects within a home to deliver them to the end user. Earlier attempts to track down objects using RFID have proven successful. MyungSik Kim and Nak Young Chong in RFID-based Mobile Robot Guidance to a Stationary Target and Direction Sensing RFID Reader for Mobile Robot Navigation developed an RFID system that "...reads the transponder-encoded data and simultaneously picks up the direction of the transponder using the received signal strength pattern."[1][2] Similarly, M.S. Kim and N.Y. Chong published a paper exploring the feasibility of having a robot with RFID navigation that could recharge itself as to have potentially unlimited active uptime. However, in Automated Robot Docking Using Direction Sensing RFID they state that “…it has not yet become possible to attain high accuracy in those techniques, particularly in cluttered or dynamically changing environments."[3] By separating the RFID navigation and object avoidance systems, we remedied this issue.

Acquiring notes from Design Issues for Assistive Robots for the Elderly by Q. Meng and M.H. Lee, they state the key issue as: “…user acceptability and this paper explores how seemingly difficult and possibly conflicting design requirements can be integrated in a human-centered approach.”[4] Taking this paper’s results into consideration, we designed the user interface to be as simplistic as possible, yet robust with features.
2. PROBLEM IDENTIFICATION

Limited mobility is a common problem in our society. Due to motor impairment, simple movement around the house can become a danger. Far too often you hear of an elderly member of society falling and injuring their hip or leg. To help prevent this, we designed a robotic assistant to help persons of disability who may have impaired motor functions retrieve objects within the house. On top of that, the robot is able to navigate the house on its own, avoiding obstacles that may stand in its path. During this project, we hope to elongate the time that the persons of disability may live on their own while simultaneously improving the field of autonomous robotics.

3. OUR APPROACH: EFFICIENT AUTONOMOUS GEOMETRIC LOCATION (EAGL) ALGORITHM

For this project, we have selected an Arduino-based IDE robot for its flexibility and scalability of size. There are currently thousands of robotic projects based around Arduino, and most are interchangeable between other Arduino-based projects. Currently, there are no designs that encompass every aspect of this research. Due to this, we have had to develop our own methods.

The method we have created is known as the Efficient Autonomous Geometric Location Algorithm or EAGL Algorithm. The EAGL Algorithm is a multi-component software algorithm. It takes elements of the object avoidance system, RFID tracking system, and battery life system into account.

The object avoidance system uses an arrangement of PING))) sensors to allow the robot to orientate itself with its surroundings. The robot will be able to, in real time, detect and avoid obstacles that fall within the acceptable range which can be changed by the end user. For example, if the robot approaches an object that falls in front of both its front and right side ultrasonic sensor, the robot will begin to rotate in place counter-clockwise until the front ultrasonic sensor receives a reading of 255, which represents an empty space, and then will continue moving forward.

The RFID tracking system is comprised of an RFID transmission module, a custom built beam antenna, and RFID transponders. The transponders are attached to different objects in the user’s home, and emit a wireless signal that carries an identification code, signaling what object it is. The beam antenna “sweeps” looking for this signal, and use the RFID transmission module to decode airwaves containing the signals.

The battery life system is a monitoring system for the robot to conserve battery life during down times, and alert the user if the aid requires a recharge before continuing on.

Figure 1: The Robotic Aid discovering which quadrant the object is located in.
The robot receives an input from the user requesting a certain object to be retrieved; the robot will begin scanning the airwaves with the beam antenna looking for the signal. If the signal is not detected, the aid begins navigating around the room, attempting to achieve a lock on the RFID tag as shown in Figure 1. Once the robot acquires the signal, the aid stops, and it begins making decisions to guide itself to the object. While keeping the object avoidance protocol running, the robot divides whichever of the 4 quadrants it detects the RFID signal in, in half. These two halves are labeled theta (ϴ) and gamma (γ) as shown in Figure 2. The algorithm then compares theta and gamma, and decides, based on signal strength, which one to navigate towards. After navigating towards the signal for a set amount of time, the protocol scans, cuts, and compares again as shown in Figure 3. This continues until the robot is upon the object and then can acquire the object to return to the user. The entire process will incorporate all systems of the EAGL Algorithm while continuously reporting to the user. Figure 4 illustrates pseudo-code for the algorithm.

**Figure 2:** The Robotic Aid dissecting the quadrant and comparing Θ and γ for higher signal strength.

**Figure 3:** The Robotic Aid further dissecting the quadrant to examine signal strength and navigate towards the object.

**Figure 4:** Pseudo-code for EAGL algorithm.

```
Initiate_system( ); //This starts the EAGL Algorithm on the Robotic Aid
Check_battery_life( ); //This checks the battery life of the aid and relays the information to the user
Wait_for_input( );
    if(true) (Check_Retrieval = true)
        //This is the wait process for accepting a job from a user
    Check_retrieval( ); //This asks the aid is current in the process of locating an object
        if (true) (RFID_scan = (true)
            else (Initiate_powersave)
        //Initiate_powersave shuts down non-essential systems while not moving such as sensors
    RFID_scan( ); //This starts the RFID antenna up to look for ID tags
    RFID_location( ); //This starts once a desired signal is located, and begins the navigation process
    Check_sensors( ); //This is for wall and collision avoidance
```
3.1 IMPLEMENTATION

The robotic aids are six legged robots, also known as hexapods. These hexapods are composed of Dynamixel AX-18 actuators, PING))) ultrasonic sensors, a ATMega644p microcontroller board, RFID Module for Arduino, a Lithium-Polymer Battery, and a custom hexapod bracket kit.

The Dynamixel AX-18 robot actuators are produced by Robotis. The actuators have a holding torque of 18 kg*cm and a max current of 2000mA\(^4\), thus providing the robot with maximum carry weight of around 5 pounds. The actuators are arranged in a "three degrees of freedom" setup, allowing movement in each robot legs to mimic that of a spider. Each leg consists of a coxa, tibia, and femur actuator.

The object avoidance system is comprised of three PING))) ultrasonic sensors. The PING))) sensors are produced by Parallax Inc. These sensors have a "view" distance from 2cm up to 3m with 20mA power consumption per sensor. The sensors measure distance using sonar. A "chirp" is transmitted from the sensor and distance to object is calculated based on time required for the "chirp" echo to return\(^5\).

The RFID transmission module is developed by SONMicro and distributed by Cooking Hacks. The radio operates at a frequency of 125KHz and has an option for secure 4Byte password transfers. The default working range is around 10cm but we will remedy this by designing a larger RFID antenna that will “sweep” the area looking for designated RFID tags.

For the most stable and reliable voltage and current, we are using a Lithium-ion Polymer battery, or LiPo. The recommended voltage for the AX-18 actuators is 11.1V with a minimum current rating of 1800mAH. Taking these requirements into consideration, we have selected an 11.1V, 2100mAH LiPo battery. The selected battery will provide more than a sufficient amount of energy for the robotic aid, while not burdening it with a large weight present in other batteries. The average battery life for each aid, without recharging, will be around 3 hours of active time with continual movement.

All of these parts are assembled in the chassis of the robot, a custom printed hexapod bracket kit. The kit has areas to place all of the materials listed, and room for further expansion as the project progresses. The chassis is composed of 1/8” ABS plastic which is ideal for this project. ABS plastics are very strong and can withstand a substantial amount of strain before reaching their tensile strength point.

6. FINAL DESIGN

![Figure 5: A demonstration of the PING))) Sensor.\(^5\)](image)

The RFID transmission module is developed by SONMicro and distributed by Cooking Hacks. The radio operates at a frequency of 125KHz and has an option for secure 4Byte password transfers. The default working range is around 10cm but we will remedy this by designing a larger RFID antenna that will “sweep” the area looking for designated RFID tags.

For the most stable and reliable voltage and current, we are using a Lithium-ion Polymer battery, or LiPo. The recommended voltage for the AX-18 actuators is 11.1V with a minimum current rating of 1800mAH. Taking these requirements into consideration, we have selected an 11.1V, 2100mAH LiPo battery. The selected battery will provide more than a sufficient amount of energy for the robotic aid, while not burdening it with a large weight present in other batteries. The average battery life for each aid, without recharging, will be around 3 hours of active time with continual movement.

All of these parts are assembled in the chassis of the robot, a custom printed hexapod bracket kit. The kit has areas to place all of the materials listed, and room for further expansion as the project progresses. The chassis is composed of 1/8” ABS plastic which is ideal for this project. ABS plastics are very strong and can withstand a substantial amount of strain before reaching their tensile strength point.

![Figure 6: The final design of the Robotic Aid](image)
After many months of trials, testing, redesigning and testing again, this is what we deemed the most efficient design. The robot is a Hexapod comprised of eighteen AX-18A servos and has a five-degree-of-freedom arm comprised of five AX-12A servos. The AX-18As were chosen for the final design for their load-bearing and the speed at which they can operate. The AX-12As were chosen for the arm because they are more cost effective than the AX-18As when speed is not a factor. There are three PING))) ultrasonic sensors on the front of the robot for ideal object avoidance when moving forward. The PING))) sensors have a large operating area so that the robot may be put into a medium-large sized room and operate effectively. The robot is powered by an ATMEGA644p board named “arbotiX”. The arbotiX is a very robust ATMEGA board and boasts many features normal boards do not such as: native AX-bus support, integrated UART connector, and a dedicated serial line for the AX servos.

7. APPLICATIONS AND FUTURE WORK

The target customers for this project are people with limited mobility, regardless of age. This Robotic Aid remedies the need for its user to bend over to pick up items or even move from the position that they are currently in. This drastically reduces the chances of the user hurting their back or possibly falling down. Furthermore, users who have trouble moving at all can still get items from around their home.

In the future, we would like to create a “dock” for the Robotic Aids so that they could recharge their battery and deposit data without the user’s input. We would also like to use slightly different sensors as the PING))) sensors have large “deadzones”. During testing we noticed that queuing up more than one object caused the Aid to become confused and disoriented. Finally, we would like to provide the robot the ability to climb stairs in the user’s home, as they are very commonplace. The long term goals of this project include an all-in-one robot aid that can do simple household chores such as collecting mail, recharging itself, navigating sidewalks, and climbing stairs.

9. CONCLUSION

At the end of this project, the created Robotic Aid successfully identifies and fetches two objects declared as “Set of Keys” and “Pill Bottle”. The original testing of the Aid showed that the fetch time of the robot was very slow and that the grabbing was not always accurate. Through trials, the speed of the fetching has improved greatly and now is within acceptable means. The average time for fetching at the beginning of the project was 5 minutes and 43 seconds with a 76% success rate. At the end of the project, we have an average fetch time of 3 minutes and 39 seconds with an 84% successful fetch rate. The main changes between early and final algorithm code was the way the robotic aid handled objects in the surrounding environments. This consisted of making decisions to navigate around an object or simply avoid it.

<table>
<thead>
<tr>
<th>Before EAGL Improvement</th>
<th>After EAGL Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Fetch Time</td>
<td>5:43</td>
</tr>
<tr>
<td>Success Rate</td>
<td>76%</td>
</tr>
</tbody>
</table>

In conclusion, the project was completed successfully. We designed and constructed a robotic aid, developed a new algorithm to search for RFID signals, implemented and tested new hardware and software, and created human-centric software that incorporates the EAGL Algorithm. The Robot Aid is a Dynamixel AX-18A based
robot with 3 degree-of-freedom legs powered by an ATMEGA644p microcontroller. The Aid also has an advanced RFID seeking arm that successfully locates and picks up objects. The Aid is very modular and can be used in many different situations.

The field of robotic aids for persons of disability has been expanding exponentially for the last decade due to breakthroughs in technology. These breakthroughs have made the field a vast and very practical topic of research. Beyond homes, these robots could provide substantial relief for workers in office, construction or even medical environments thanks to the scalability. The purpose of this project was to make homes a safer and easier place for disabled people to live while relying on robots for aid.

ACKNOWLEDGEMENT

This work was supported by a grant from the NASA WV Space Grant Consortium and Shepherd University. I would like to thank Dr. Kim for his guidance and encouragement during this project and the Department of Computer, Mathematics and Engineering for the opportunity to expand this project beyond ideas and drawings. I would also like to thank President Susan Shipley for funding of the research and continued support of the research fellowship.

REFERENCES

The World through Robotic Eyes:
Developing Boosted Classifiers for Robotic Object Recognition with OpenCV
Courtney Crites

Department of Computer Sciences, Mathematics, and Engineering
Shepherd University
P.O. Box 3210
Shepherdstown, WV 25443
c bitrate03@shepherd.edu

ABSTRACT

The field of domestic service robotics is an extremely useful and rapidly growing field that focuses on robots with the ability to assist human-beings in their homes. This assistance can range anywhere from lending a hand with everyday chores to helping the elderly and disabled to live on their own. The latter, in particular, is becoming more important with each passing day. A robot that assists people with the things they are incapable of doing could be an invaluable asset. However, we must also resolve the programming problems that come with such a robot. One of the primary issues is the recognition of objects. If a robot is going to help humans it needs to know what it is looking at before it tries to do anything. The robot needs to be able to see the physical object, identify the object, and if necessary, report what the object is. A robot with this process as its primary function would actually be a great asset to people who are visually impaired. The robot would be a seeing-eye dog with much greater potential: the potential to both speak and physically act upon the object in a human-like way. The motivation behind this research is to improve existing recognition methods, and aid in the eventual development of a robotic device that can recognize objects quickly and accurately enough to be of use in real-world applications. In this paper, a brief overview is given of object detection and recognition methods in robotics. After that, focus will switch to OpenCV and its own integrated methods for object recognition. While it is not as accurate as some methods, it is an excellent tool for real time analysis and recognition. For this research project, Microsoft Visual Studio 2010 and OpenCV (Open Source Computer Vision) libraries are used in conjunction to create a C++ program with the ability to detect, recognize, and track the location of several objects using a feed from a camera. During this process, ways to improve the detection precision and accuracy will be developed and tested. It is then hoped that this program can be integrated into a robotic system.

KEYWORDS: ROBOT, OPENCV, OBJECT RECOGNITION, HAAR-LIKE FEATURES, ADABOOST ALGORITHM

1. PROBLEM

Object detection and recognition have long been major challenges in the world of computer vision. For sake of clarity, detection must be differentiated from recognition. Detection is simply recognizing that an object is present. Recognition is the ability to know what the object is. The goal of this research is to use Visual Studio and the Open Source Computer Vision libraries to create a C++ program for object recognition. Four different object classifiers
are created for object recognition. These four include a fork, a butter knife, a glue bottle, and a water bottle. During this task, ways to improve the existing methods of training and recognition are developed and tested.

2. INTRODUCTION TO OBJECT RECOGNITION

In order for a robot to interact with its environment, it must first be able to see its environment. However, it cannot optically view the world on the level of a human. The average adult human can look at a small assortment of simple household items, and quickly determine what each one is and how it is used. The most important tool in this scenario is the human’s brain. The brain is essentially a massive database of all the information the human has gathered in his or her lifetime. Additionally, we can see a broad range of items at one time, and all the 2D and 3D elements of those objects. Our ability to perceive the same object from multiple angles is also important [M. Bjorkman, 2006]. Applying these human visual skills to robotics is not a simple task. How does the robot differentiate one object from another, and subsequently define what it is? How does the robot recognize the same thing from multiple perspectives? The answer is all in the programming. A robot must be programmed to know what objects look like and what they do not look like. However, programming like this requires a bit of learning on the robot’s part. The ways to accomplish this learning process are outlined in the early parts of this paper. While object recognition involves some heavy mathematics and advanced algorithms, these sections will primarily focus on the theories, ideas, and applications behind the math. In later sections, there will be an emphasis on the Open Source Computer Vision, libraries and the use of haar-like features in object recognition.

3. THE DIFFERENT FORMS OF OBJECT RECOGNITION

When first approaching the topic of object recognition, one must consider both 2-dimensional and 3-dimensional objects. 2D recognition has already reached the stage where it can be used successfully in real-world applications. 3D recognition, however, is still being developed toward that goal. It has not been as heavily worked with and it is filled with complex programming and hardware problems to solve. What clearly differentiates 3D from 2D is that 3D is much more complex from a visual standpoint. Though it may be obvious, it must be pointed out that there is quite a bit more information to a 3D object. 2D object recognition methods only have to solve the problems of translation and rotation. 3D objects, however, can be viewed from an infinite number of angles, and may look completely different turned one way than they do another way. Recognition methods must be devised that are capable of understanding and processing such a large wealth of information [P. Viola, 2001].

The type of hardware used can also play a role in how an object is recognized. Sensors can be either contact or non-contact. Contact sensors physically touch and measure the object. Non-contact sensors, of course, do not touch the object. These include range sensors and camera setups. Non-contact sensing is preferred in industry since there is little to no chance of damaging an object, and it is less expensive and quicker than using moving parts with touch sensors [P. Viola, 2001].

There are also two types of vision systems: active and passive. In active vision, a light with a pattern is projected onto the object. Then, a camera gets an image of the
object with the projected pattern. It can be quite useful with 3D objects, since the pattern will show contours, edges, and distance on the object. However, it is also a more expensive method. The passive vision method uses one or more cameras to get images of an object, and combines that with an algorithm for object recognition. This paper will focus almost entirely on non-contact, passive vision systems due to their lower expense and the greater abundance of existing research [P. Viola, 2001].

When it comes to the actual identification methods of the objects, there are two approaches that can be taken. Some methods are generative, and others are discriminative. Generative models estimate the likelihood of an object existing, and then the object is assigned the most likely class. These models search to find a representation of the original data. Discriminative models discriminate in what is being looked at in order to find the object of interest. The purpose of these models is primarily for classification tasks. Using data from training, these methods try to reach optimal decision boundaries. When looking at an unknown sample, a decision boundary is estimated, and a corresponding label is assigned to the sample [P. Roth, 2008].

In object recognition, there are three main categories of features used for identification. First are appearance based features. These are usually taken from 2D images of the object of interest. Examples include color shade and intensity. The second category is shape-based. These features represent the object by shape and contour. Third are model based features. Objects are approximately represented as basic 3D geometric shapes. These shapes include boxes, spheres, cones, cylinders, generalized cylinders, and surface of revolution [P. Roth, 2008].

Each method of recognition can be split into two main classes: local and global. A local feature is a particular detail located in a very small region, including even pixel-sized regions. It is a single piece of information distinctive to the object of interest. Local features include color, average gradient, and average gray values of pixels or small regions. When doing object recognition, local features should not be affected by illumination changes, noise, scale changes, or viewing direction. However, this is often not possible due to how simple these single features are. Therefore, several features of a single point are combined into a more complex description. This new image description is normally referred to as a descriptor [P. Roth, 2008].

Global features, on the other hand, look at the image as a whole. They try to consider all the pixels in the image. If necessary, the original image of the object can be reconstructed. This means, in some cases, object recognition can be very accurate. However when it comes to objects that may only feature partially in an image, local features have the advantage. They can still recognize an object even if only a small part of it is actually visible [P. Roth, 2008].

4. OPENCV & HARR-LIKE FEATURES

One of the more prominent and well-developed ways to create object recognition programs is through use of OpenCV, or Open Source Computer Vision. OpenCV is a library of over 500 programming functions for the purpose of real-time computer vision. The library can be used with the programming languages C, C++, and Python. One of the major advantages of OpenCV recognition is its near-instantaneous speed of recognition, making it extremely useful in real-world applications [OpenCV Wiki]. In OpenCV, classifiers for recognizing different objects
must be created by the user. An OpenCV installation already comes with a trained classifier for frontal face direction, but it is not limited only to that. Just about any object can be recognized if there is a classifier created for it. These classifiers depend on local features which are based on haar functions. These features are referred to as haar-like features.

Haar-like features are attributes found in images when looking at the intensities of pixel shades across areas. Several groups have used them for the purpose of object recognition in order to quickly find particular objects in an image [C. Papageorgiou, 1998i][P. Roth, 2008]. Figure 1 shows the most basic attributes and shapes of these features. In a study done by Viola & Jones, a new, faster method of image representation is presented, called an integral image, and it is then used to rapidly find haar-like features within a given image. The study says that “an integral image can be computed from an image using a few operations per pixel. Once computed, any one of these haar-like features can be computed at any scale or location in constant time” [P. Viola, 2001] . Real time calculation is important since many applications of object recognition have to be able to quickly and accurately recognize objects as soon as they see them. It is inconvenient to take pictures or video, and then scan the media piece by piece in order to find the object of interest. In a good robotic object recognition system, the looking and the recognizing should be occurring simultaneously.

Figure 1. Four different categories of Haar-like features, including 45° tilted versions added by Lienhart & Maydt[R. Lienhart].

The features depicted in Figure 1 can be separated into different groups based on the number of rectangles involved. Two-rectangle features, seen in category 1 above, are the difference between of the sum of the pixels in two horizontally or vertically adjacent rectangles placed on the image. These rectangles are the same size and shape. Category 3 also includes two-rectangle features, but they are the difference between the sum of pixels of one larger rectangle and a smaller one centered inside it. Three-rectangle features, shown in category 2 above, compute the sum within two outside rectangles subtracted from the sum of a center rectangle. Then there is the four-rectangle feature, shown at the bottom of Figure 1. It computes the difference between diagonal pairs of rectangles [P. Viola, 2001]. Originally, the tilted haar-like features were not part of the basic set until they were tested and added by Lienhart & Maydt. The extended feature set
complicated the learning process somewhat, but greatly improved recognition accuracy. The extended set produced a 10% lower false alarm rate when tested with face recognition. [R. Lienhart].

The problem with calculating these features is the sheer number of them that exist in an image. Therefore, an algorithm called AdaBoost (Adaptive Boosting) exists to train “weak” object recognition classifiers into stronger ones by using the most heavily reoccurring haar-like features in the set with the lowest chance of error. Every round of boosting selects the most-likely feature out of the possible 180,000 features in an image sub-window, and gives it more weight during a search. These features are essentially strung together from most common to least common, forming what is known as a classifier cascade [P. Viola, 2001]. In a classifier cascade, if the most common feature is found, then the program begins to search for the second most common feature. If that is found, it moves on to the third. This process repeats for a certain threshold level. At this point, the area containing these features is marked as an object-containing area.

![Figure 2. Examples of common haar-like features found during facial recognition [P. Viola, 2001].](image)

Figure 2 shows how faces can be recognized using just a few specific haar-like features. The two features shown above are the first and second most likely facial features picked by the Adaboost algorithm. The top row of Figure 2 shows only the features, and the second row shows the features overlaid in the corresponding areas on a face. As one may observe with most faces, the area across the eyes is much darker than the area across the upper cheeks. It can also be observed that the eye regions are darker than the bridge of the nose. These facial features are clearly demonstrated by the simplified rectangular features [P. Viola, 2001].

5. PROCEDURE

For this research project, OpenCV and Visual Studio 2010 were installed onto a laptop with a 2.1 GHz dual-core processor and 4 GB of RAM. The camera used was a built-in 1.3 megapixel camera on the laptop.

The first step in classifier training is the collection of images. Two main sets of pictures are collected: negative and positive. A negative set includes photos that do not have the object of interest, while a positive set includes images that do have the object. There should be 2,000 to 5,000 negative samples for a good classifier. However, for this research, small negative data sets of 500-1,000 were used to save time during training. There are two options for positive sets. One can make positive samples from a few images, or positive samples from one image with distortions.

Once all images are gathered, all negative image filenames must be listed in a text file. After this, the generated positive image samples are packed into a vector file as thumbnails. Finally, haartraining.exe is run to compare the positive and negative images, pick out the most common haar-like features of the object, and test them accuracy. Multiple stages of adaboosting run in order to create several strong classifiers, and then these strong classifiers are
combined into an XML file to create a cascade of boosted classifiers.

After the classifier is created, it can be placed into the program. The program’s main purpose is to search the frames of a camera feed for the object, correctly identify the object when it is present, and track the objects location.

6. OUR APPROACH: POSITIVE SAMPLE GENERATION ALGORITHM

While use of haar-like features is a powerful method for object recognition, there is still a lot of room for improvement. Therefore, steps were taken to improve both the training and performance of the classifiers. First of all, the haartraining application provided by OpenCV suffered many crashes during training due to problems with memory and possible OS incompatibility. Therefore, it was rebuilt from the source code using Visual Studio. Furthermore, since the original application only utilized one thread, a VS-compatible API called OpenMP was used to enable multi-threading. With just a few lines of new code, the haartraining program was able to run with four simultaneous threads [MSDN].

With the optimization problems out of the way, a method was developed that would increase recognition accuracy and simplify the positive sample collection process. Creating samples from one image meant there was a lack of variety, and gathering many positive images was a time-consuming task. Therefore, multiple unique vector files were generated for several different positive images, and then those files were merged together into one large vector file. The vector merging application used in this project was developed by Naotoshi Seo[8]. 100 samples were generated from each single image, and 5-10 unique images were used. This led to 500-1000 positive samples for each object’s data set.

**PSEUDOCODE**

//begin with a collection of images of object of interest on black background

```cpp
for(int x=0; x<=inputImgNum; x++) //repeats process for number of initial images
{
    PositiveSamplePrep(inputImg){ //Takes an input image to process it
        image imageN =inputImg;
        image imgGray, imgThresh, ObjectImg;
        int ThreshNum, TargetSampleNum;
        VEC OutputVEC;

        conversionGrayscale(imageN) /*converts image to grayscale*/
        {
            imgGray= convertGray(imageN)
            return imgGray;
        }

        imageBinaryThresh(ThreshNum, imageN) /*thresholds grayscale image with user-entered value*/
        {
            imgThresh=thresh(imageN, ThreshNum);
            return imgThresh;
        }

        ImageGray-ImageThresh= ObjectImg; /*removes 0(black) threshold image areas from the respective grayscale image areas, object img left*/

        ObjectImgBackgroundOverlay(ObjectImg, TargetSampleNum) /*overlays object onto backgrounds*/
        {
            image[] finalImgs; //creates a grouping of final images
            for(int num=0, num<=TargetSampleNum, num++) /*repeats process for desired number of samples*/
            {
                image Bg=random.background; /*selects random backgrounds from negative dataset*/
                image OverlayImg= Overlay(ObjectImg, Bg);
                finalImgs[num]= OverlayImg
            }
            return finalImgs;
        }

        VECGen(finalImgs, OutputVEC) /*takes group of final images, converts to VEC file*/
        {
            VEC(finalImgs, OutputVEC)
            return OutputVEC;
        }

        FinalVec= FinalVec + OutputVec; /*merges output VEC files into a final VEC file*/
    }
}
save FinalVec; /*saves final VEC file to disc for classifier training purposes*/

Figure 4. Samples generated from one image using the method described. Objects in this set were also randomly offset at slightly different angles.

There are, of course, a few problems with this method. The most obvious, perhaps, is the recognition of objects with dark or black areas. You can see in Figure 4 that a small portion of the bottle label was removed. While this has very little effect on the recognition in this particular case, it can easily become a problem with different sorts of objects. It should be pointed out that while there are certainly better ways to crop out the edges of an image, the key to this process is the automation of such methods. Individually editing all samples by hand can quickly become tedious and difficult. Thresholding offers a quick and acceptable way to accomplish the task.

7. RESULTS

During this research, multiple classifiers were created for a wide range of objects. The most notable examples included a fork, a butter knife, a glue bottle, and a water bottle. The training and testing of these classifiers made it apparent that the use of haar-like features can often be a hit or miss method. Furthermore, during creation of these classifiers, a few major cons became apparent. First of all, the material of the object can be a big obstacle. A prominent example is metal forks and knives. In training images, the lighting on forks is flat and static. In real-time situations, however, lighting is dynamic and constantly changing. It also varies in intensity. This causes failure of the program to recognize metal objects. However, if plastic is used, recognition accuracy in the video feed increases by a notable amount. This was seen using both a metal silver-colored fork and a plastic white fork on the same classifier. The metal fork was undetected, but the plastic fork was recognized easily. The butter knife was also a problem. Figure 6 shown below demonstrates the lighting variations on the metal butter knife used in testing, which led to poor recognition.

Figure 5. Examples of positive butter knife samples demonstrating how different one object can appear in the real world.

Lighting was not a problem exclusive to shiny objects, however. Figure 7 below shows how good recognition of a glue bottle can quickly become poor recognition with a
simple lighting change. It is likely that this is due to the reduced contrast in the image.

**Figure 7.** The same classifier and program viewing the exact same area with the lights off and then on.

The second problem encountered was the orientation of objects. Classifiers using haar-like features can only be trained to recognize one orientation of an object at a time. Mixing positive object images of different orientations can be done, but it will lead to confusion during classification, causing false positives and lowered accuracy. To get around this, multiple classifiers can be trained for a few main orientations of an object. However, it is still a tedious process. In this experiment, only one vertical position was trained for each object. In the future, extra classifiers will be trained for other object positions.

Figures 8, 9, and 10 show the final results achieved from the four different objects. Although it was just stated that only vertical classifiers were trained, a fairly successful diagonal fork detector emerged from early training experiments and is displayed in image A and B in the set. Image A also shows attempts to confuse the classifier with other objects. The yellow wires were incorrectly identified as a fork, and represent a false positive.

**Figure 8.** Final Output of the Classifiers for a plastic fork, butter knife, and a bottle of glue.

**Figure 9.** Final output of glue & wheel classifiers, and their combination with the pencil classifier.
Figure 10. Full output of bottle recognition, with top left pixel coordinates on left for every instance of recognition. About 6 previous false positives can be seen in the coordinate data.

Figure 11. Two separate runs of the program using a classifier created with the basic method, and a classifier created with our method. The basic method gives a detection rate of about 1.5 detections per
second. Our method of positive sample generation gives results of 3 detections per second. This speed up in detection is an indicator of increased accuracy, as the classifier has a better understanding of what a glue bottle is supposed to look like in the real world.

8. APPLICATIONS & FUTURE WORK

The field has countless applications, making it an extremely deep and useful topic of research. These applications include, but are not limited to, product manufacturing and inspection, counting and measuring, robotic surgery, object modeling, surveillance, security, mapping, and navigation [S. Xie, 2008]. The purpose of this research is to find a reliable object recognizing method that can be implemented into a robotic system for use in the real world.

Future research will be conducted with real-time object recognition through use of other OpenCV methods. Examples of these other methods include Local Binary Patterns and SURF (Speeded Up Robust Features). Both of these methods focus on local features of objects, and may offer greater accuracy not available with haar-like features. Use of these methods, and others, will also be implemented into robotic systems in order to begin developing real-world applications.

9. CONCLUSION

Throughout this research, four classifiers were created, but only two of them, the fork and the glue bottle classifiers, accurately recognized their object with enough frequency to be of use. Even then, they are not yet accurate enough for any real-time applications. Further training and development is needed to improve their usefulness. Further work also needs to be done on integrating the different classifiers into one program. Currently, using more than one classifier at a time leads to slowdown and missed instances of recognition. Solving these problems is vital for the creation of an object-recognizing robot.

REFERENCES


The Mechanical Properties of a Langmuir Monolayer Made of a Straight Alkane

Compared to Those Made of Straight Chain Fatty Acids

Gerren D. Simms and Dr. Mengyang Li

Department of Computer Sciences, Mathematics, and Engineering
Shepherd University
P.O. Box 3210
Shepherdstown, WV 25443
Gsimms01@rams.shepherd.edu, Mli@shepherd.edu

ABSTRACT
Traditionally Langmuir monolayers and lipids bilayers (including cell membranes) on water interfaces are made of amphiphilic molecules with both polar hydrophilic and non-polar hydrophobic parts. Since the discovery of a Langmuir monolayer of a non-polar hydrophobic only fluorocarbon chain molecule on water by the advisor of this project[1], other groups have found that non-polar hydrophobic only long-chain hydrocarbons (n-alkanes) with chain length greater than 30 carbons can also form Langmuir monolayers on water[2]. The lateral compression mechanical properties of a Langmuir monolayer of a non-polar hydrophobic only hydrocarbon chain molecule of 35 carbons (Pentatriacontane) was researched and then compared to past research at Shepherd University, which studied the mechanical properties of Langmuir monolayers of straight chain fatty acids which all have a polar hydrophilic head group.

KEYWORDS: Langmuir, Monolayer, Hydrophobic, Pentatriacontane, C35H72

1. INTRODUCTION
Langmuir monolayers are single layers of surfactant molecules at liquid interfaces. Studying the properties of these surfactant molecules in a two-dimensional array at the interface and under surface pressure helps to further understand the chemical, physical, and biological processes that occur at interfaces. Information about phase diagrams, phase transitions, and mechanical properties of Langmuir monolayers can be obtained by measuring and analyzing their surface pressure vs. surface area isotherms. The data obtained during experimentation on pentatriacontane is being compared to previous research on straight chain fatty acids.

2. BACKGROUND
Before the mechanical properties of a Langmuir monolayer can be studied, confirmation that a monolayer has indeed been formed is required. It has been confirmed, through x-ray diffraction, that during the lateral compression of a Langmuir monolayer, the maximum modulus will be found in an area around 19 Å²/molecule[3]. The maximum modulus is the film's resistance to lateral compression and can be described as the point with greatest slope on the surface pressure vs. surface area isotherms. The maximum modulus can also be described using the following equation: $K^{-1} = -A \left(\frac{d\pi}{dA}\right)$.
Where $K^{-1}$ is the modulus at a specific point, $A$ is the area at the same point, and $\pi$ is the surface pressure at the point.

Pentatriacontane $\text{H(CH}_2\text{)}_{35}\text{H}$ straight chain alkane ($C_{35}$ alkane), was used to form monolayers and then the mechanical properties were measured and compared to the straight chain fatty acids of lengths $C_{24}$ and $C_{18}$. The straight chain fatty acid data for $C_{24}$ and $C_{18}$ was collected by students, Brittanii Love, Kelli Moffett, Phillip Sinsky, and Britanny Ioset under the guidance of Dr. Mengyang Li.

3. RESULTS

Monolayers were formed using the $C_{35}$ alkane at three different compression rates. These results were verified by calculating the average area in which the maximum modulus was found.

<table>
<thead>
<tr>
<th>Compression Rate (Å²/molecule/min)</th>
<th>Average Area (Å²/molecule)</th>
<th>Average Area (Å²/molecule)</th>
</tr>
</thead>
<tbody>
<tr>
<td>41.498</td>
<td>19.185 ± 2.190</td>
<td></td>
</tr>
<tr>
<td>27.665</td>
<td>17.003 ± 1.311</td>
<td></td>
</tr>
<tr>
<td>3.654</td>
<td>21.062 ± 1.864</td>
<td></td>
</tr>
</tbody>
</table>

The results shown in Table 1 confirm that a monolayer has been formed because the area in which the maximum modulus was detected exists in the approximate range of 19 Å²/molecule.

Not only did the $C_{35}$ alkane follow the same trend at all three compression rates, but it also followed the same trend as the straight chain fatty acids.
This data illuminates the fact that all three molecules being compared to each other follow the same trend in the area that the maximum modulus was found.

Another mechanical property of the monolayers that was compared was the value of the maximum modulus at different compression rates.

<table>
<thead>
<tr>
<th>Compression Rate (Å²/molecule/min)</th>
<th>Avg. Max Modulus (mN/m)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C_{35}</strong> 41.498</td>
<td>84.699± 9.441</td>
</tr>
<tr>
<td>27.665</td>
<td>91.478± 4.227</td>
</tr>
<tr>
<td>3.654</td>
<td>244.210± 15.270</td>
</tr>
<tr>
<td><strong>C_{24}</strong> 44.704</td>
<td>292.870± 56.109</td>
</tr>
<tr>
<td>26.822</td>
<td>437.562± 126.249</td>
</tr>
<tr>
<td>6.096</td>
<td>470.899± 126.861</td>
</tr>
<tr>
<td><strong>C_{18}</strong> 44.963</td>
<td>364.434± 111.621</td>
</tr>
<tr>
<td>26.978</td>
<td>411.361± 138.422</td>
</tr>
<tr>
<td>6.131</td>
<td>628.622± 173.357</td>
</tr>
</tbody>
</table>
The general trend discovered was that as the compression rate decreased, the resistance to lateral compression (maximum modulus) increased, although the value of the maximum modulus for the alkane was approximately half of that of the fatty acids’.

4. DISCUSSION

The overall goal of this project is to continue the furthering of monolayer research and its many applications. Monolayer research is considered theoretical, but has had many applications in the study of chemical, physical, and biological processes at interfaces. Some examples are the heterogeneous catalysis of chemical reactions, physical and chemical adsorptions, and cell membrane processes.

This research shows that regardless of whether a molecule is amphiphilic or hydrophobic, once a monolayer is formed, the mechanical properties of that monolayer will follow some general trends. One of the most notable trends is that when the compression rate of the monolayer is decreased, the resistance to lateral compression is increased.

The fact that the C35 alkanes’ resistance was approximately half of that of the fatty acids could be due to a few circumstances. Some would say that it is possible that the amphiphilic nature of the fatty acids can increase their resistance to lateral compression, but most likely this is due to the length of the molecule itself. To best exploit this difference, a C35 fatty acid molecule should be obtained and then once a monolayer is formed, the mechanical properties can be studied and then compared to the C35 alkane.

The overall accomplishments of this research would be considered to be
successful. It appears that this is the first monolayer to be formed and/or studied using the C₃₅ alkane molecule. The research also helps to reconfirm that whether a molecule is amphiphilic or hydrophobic, it will not only form a monolayer, but it will also follow the same general trends in its mechanical properties.

ACKNOWLEDGEMENT
This work was supported by a grant from the NASA WV Space Grant Consortium, and Shepherd University, with special thanks going Dr. Kim for his logistical support to the establishment of this research project. An expression of thanks is also in order for Dr. Mengyang Li and his work on behalf of this project and the related literary technical works made possible through the NASA WV Space Grant Consortium. Dr. Li has been kind enough to pass along a sliver of his wealth of knowledge in support of this project. Also a special thanks to President Shipley, without whom, this research effort would not have been possible.

REFERENCES

