FACULDADE DE TECNOLOGIA DE SÃO JOSÉ DOS CAMPOS
FATEC PROFESSOR JESSEN VIDAL

GUSTAVO GUIMARÃES FARIA

MAGIC SURFACE: INTERFACE WEB PARA APLICATIVO DE REALIDADE AUMENTADA

São José dos Campos
2015

GUSTAVO GUIMARÃES FARIA

MAGIC SURFACE: INTERFACE WEB PARA APLICATIVO DE REALIDADE AUMENTADA

Trabalho de Graduação apresentado à Faculdade de Tecnologia São José dos Campos, como parte dos requisitos necessários para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Me. Giuliano Araujo Bertoti

São José dos Campos

2015

Dados Internacionais de Catalogação-na-Publicação (CIP)

Divisão de Informação e Documentação

FARIA, Gustavo Guimarães
Magic Surface: Interface web para aplicativo de realidade aumentada.
São José dos Campos, 2015.
51f.

Trabalho de Graduação – Curso de Tecnologia em Análise e Desenvolvimento de Sistemas, FATEC de São José dos Campos: Professor Jessen Vidal, 2015.
Orientador: Me. Giuliano Araujo Bertoti.

1. Áreas de conhecimento. I. Faculdade de Tecnologia. FATEC de São José dos Campos: Professor Jessen Vidal. Divisão de Informação e Documentação. II. Título

REFERÊNCIA BIBLIOGRÁFICA

FARIA, Gustavo Guimarães. **Magic Surface: Interface web para aplicativo de realidade aumentada.** 2015. 51f. Trabalho de Graduação - FATEC de São José dos Campos: Professor Jessen Vidal.

CESSÃO DE DIREITOS –

FARIA: Gustavo Guimarães
TÍTULO DO TRABALHO: Magic Surface: Interface web para aplicativo de realidade aumentada
TIPO DO TRABALHO/ANO: Trabalho de Graduação / 2015.

_____

Gustavo Guimarães Faria

Avenida vinte e três de dezembro, 339

CEP 12225-480 – São José dos Campos – São Paulo

Gustavo Guimarães Faria

MAGIC SURFACE: INTERFACE WEB PARA APLICATIVO DE REALIDADE AUMENTADA

Trabalho de Graduação apresentado à Faculdade de Tecnologia São José dos Campos, como parte dos requisitos necessários para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Composição da Banca

Reinaldo Gen Ichiro Arakaki, Doutor, FATEC

Giuliano Araujo Bertoti, Mestre, FATEC

Luan Rafael Castor Pinheiro, Tecnólogo, FATEC

_____/_____/_____

DATA DA APROVAÇÃO

Dedico este trabalho aos meus familiares e amigos. Sem eles, essa jornada não teria sido cumprida.

AGRADECIMENTOS

Agradeço ao professor e orientador Giuliano Bertoti, pelo apoio e incentivo a desenvolver este trabalho.

Aos professores da FATEC que contribuíram com a minha evolução durante toda a jornada do curso de Análise e Desenvolvimento de Sistemas.

Aos professores da ETEC de São José dos Campos que me ensinaram a base do desenvolvimento de software e plantaram essa semente, para que eu prossiga em busca de novos conhecimentos na área.

Aos meus amigos e companheiros de classe, Marcos Bustamante, Marcos Araújo e Samara Cardoso, pela parceria e apoio durante o curso. Sem eles, chegar ao fim teria sido muito mais difícil.

E principalmente aos meus familiares, que estiveram ao meu lado durante os momentos difíceis, e me incentivaram a nunca desistir.

“The victim should have the right to end his life, if he wants. But I think it would be a great mistake. However bad life may seem, there is always something you can do and succeed at. While there's life, there is hope”.

Stephen Hawking

RESUMO

O objetivo deste trabalho é desenvolver uma interface *web* para uma aplicação de realidade aumentada, que permita salvar imagens e vídeos baseados na localização geográfica do usuário e representar tais arquivos no mundo real, através da câmera de um dispositivo móvel. Para este fim, esse trabalho apresenta uma aplicação que busca eternizar momentos em seus locais de acontecimento, para que possam ser revividos no futuro por qualquer pessoa ao apontar a câmera do dispositivo móvel para o local. Como resultado, foi desenvolvido um aplicativo, que pode ser acessado por qualquer *smartphone* ou *tablet*, independente de sistema operacional. Com isso, quaisquer pessoas com interesses em eternizar momentos em seus locais de origem podem utilizar livremente a aplicação.

Palavras-Chave: Realidade Aumentada; Dispositivos Móveis; Interface.

ABSTRACT

The objective of this work is to develop a web interface for an application of augmented reality, which allows saving images and videos based on the geographical location of the user and to represent that files in the real world through a mobile device camera. To this end, this paper presents an application that seeks to immortalize moments in their places, so they can be revived in the future by anyone by pointing the camera of the mobile device to the place. As a result, an application that can be accessed by any smartphone or tablet, regardless of operating system was developed. Thus, any person with an interest in perpetuating moments in their origin places can freely use the application.

Key-Words: Augmented Reality; Mobile Devices; Interface.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

MIT	*Massachussets Institute of Technology*
HTML	*Hypertext Markup Language*
GPS	*Global Positioning System*
API	*Application Programming Interface*
RA	Realidade Aumentada
RV	Realidade Virtual
CLI	*Command Line Interface*
GUI	*Graphical User Interface*
NUI	*Natural User Interface*
CSS	*Cascading Style Sheets*

SUMÁRIO

1 - INTRODUÇÃO

1.1 – Motivação

A crescente evolução dos dispositivos móveis está possibilitando a criação de aplicações cada vez mais inovadoras por oferecer diversas formas de interação com o usuário. Hoje, é possível interagir com o celular através de toques na tela, comandos de voz, gestos, entre outras formas. Este tipo de interface, é conhecida como Interface Natural, que visa adaptar os comandos baseados em ações naturais do usuário, em alguns casos, transmitindo a sensação de magia.

Segundo Clarke, qualquer tecnologia suficientemente avançada é indistinguível de magia (CLARKE, 1998). Por essa razão, pesquisadores do MIT Media Lab criaram o curso "*Indistinguishable From…Magic as Interface, Technology, and Tradition*", onde são desenvolvidos projetos tecnológicos utilizando conceitos de mágica (MIT, 2015).

Uma das áreas tecnológicas em crescimento é a Realidade Aumentada, que representa objetos virtuais no mundo real através de uma câmera, mais comum em dispositivos móveis como *smartphones* e *tablets.* Utilizando conceitos de realidade aumentada, pode-se desenvolver uma aplicação que oferece ao usuário uma experiência única de poder eternizar momentos de sua vida que ocorreram em um determinado lugar, tornando possível reviver tais momentos no futuro simplesmente apontando a câmera do dispositivo móvel para o local de origem, seja pelo próprio usuário que registrou quanto pelas gerações futuras que venham ter acesso à experiência registrada naquele lugar.

Vale ressaltar que magia não é ciência e, portanto, não é objeto de estudo deste trabalho. Neste contexto, o termo "magia" é usado apenas como metáfora para os conceitos de interação humano computador e realidade aumentada conforme estudos do MIT Media Lab.

1.2 - Objetivos

As subseções a seguir apresentam os objetivos deste trabalho.

1.2.1 - Objetivo Geral

Este trabalho de graduação tem por objetivo desenvolver uma interface web para uma aplicação de realidade aumentada que permita registrar imagens e vídeos no mundo real através de *smartphones* e *tablets*.

1.2.2 - Objetivos Específicos

Os objetivos específicos deste trabalho são:

- Desenvolver uma interface web *Mobile Only* para melhor usabilidade em dispositivos móveis;
- Utilizar sensores de localização, giroscópio e câmera para aplicar os conceitos de realidade aumentada;
- Utilizar um *Web Service* para registro e controle dos dados;
- Disponibilizar a aplicação para a sociedade.

1.3 - Metodologia

Para o desenvolvimento deste trabalho será necessário coletar dados do GPS, acelerômetro e câmera, sensores que estão presentes nos smartphones atuais. A nova versão do *Hypertext Markup Language*, ou HTML 5, tem suporte à eventos capazes de capturar estes dados. A partir destes eventos, os dados serão enviados para um *WebService* e depois representados em um mapa gerado pela API do Google Maps, e também através da imagem da câmera em tempo real. A implementação das funcionalidades será totalmente em JavaScript, que é uma linguagem interpretada e baseada em objetos e é utilizada para dar interatividade a uma página web (MDN, 2014).

1.4 - Organização do Trabalho

Este trabalho está organizado nos seguintes capítulos:

- Capítulo 2- Revisão Bibliográfica: apresenta os conceitos e estudos relacionados a este trabalho;
- Capítulo 3- Desenvolvimento: aborda a implementação do aplicativo Magic Surface;
- Capítulo 4- Resultados: apresenta os resultados obtidos durante o desenvolvimento;
- Capítulo 5- Considerações finais: conclui e apresenta os trabalhos futuros relacionados a este trabalho.

2 - REVISÃO BIBLIOGRÁFICA

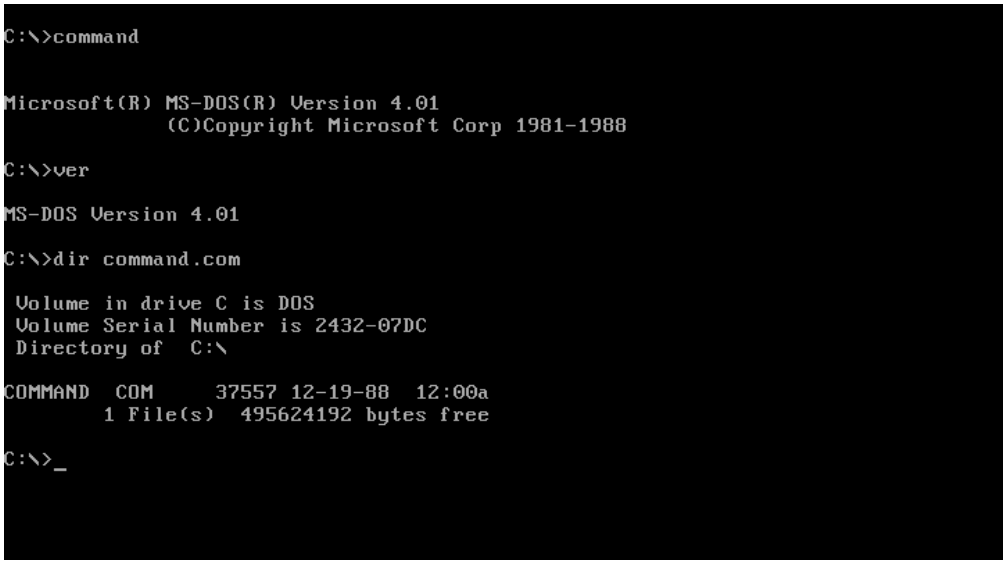
2.1 – Interface de Usuário

Interface é toda porção de um sistema com a qual o usuário mantém contato ao utilizá-lo, tanto ativa quanto passivamente. Com a evolução da tecnologia, as interfaces de usuário também evoluíram, e atualmente existem três tipos de interface, são elas: CLI – *Command Line Interface* ou Interface de Linha de Comando, GUI – *Graphical User Interface* ou Interface Gráfica de Usuário e NUI – *Natural User Interface* ou Interface Natural de Usuário.

2.1.1 – Interface de Linha de Comando

A CLI é uma interface de usuário para sistemas operacionais de computadores que disponibiliza um terminal onde o usuário digita um comando em uma linha específica, recebe uma resposta do sistema, e, em seguida, entra com outro comando, e assim por diante. O Prompt de Comando do MS-DOS em um sistema operacional Windows é um exemplo da disposição de uma interface de linha de comando.

Figura 1 - Prompt de Comando MS-DOS

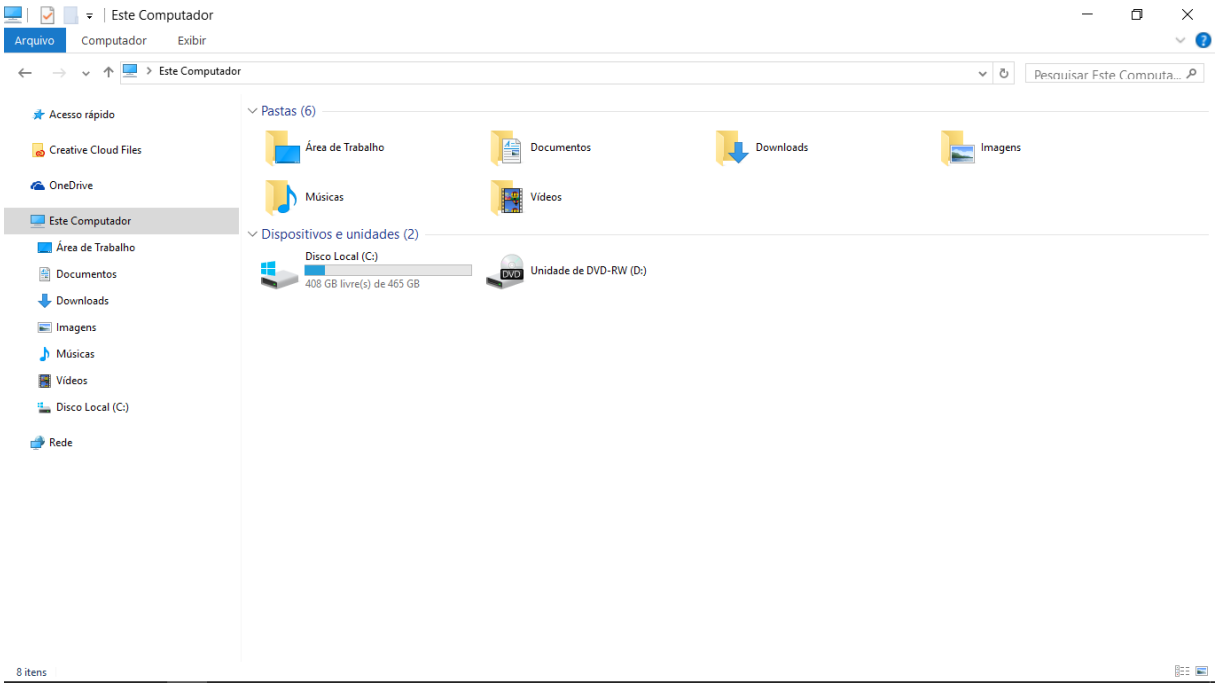


Uma das vantagens da CLI, é a produtividade, que em alguns casos, pode aumentar caso o usuário possua conhecimento avançado sobre a linguagem dos comandos. No entanto, assim surge a grande desvantagem desta interface. Um usuário sem conhecimento não consegue manipular o sistema, gerando perda de produtividade.

2.1.2 – Interface Gráfica de Usuário

É uma interface que oferece elementos gráficos, tais como janelas, ícones, menus e botões. O termo foi criado na década de 1970 para distinguir das interfaces baseadas em linhas de comando. No entanto, hoje, a GUI é a interface mais utilizada na computação. A figura abaixo, mostra um exemplo de interface gráfica de usuário. Trata-se do Explorador de Arquivos do sistema operacional Windows 10, um programa que oferece menus e ícones para manipulação de arquivos e diretórios do sistema.

Figura 2 - Interface Gráfica de Usuário



2.1.3 – Interface Natural de Usuário

A Interface Natural de Usuário ou *Natural User Interface* (NUI) é uma interface de camada invisível, utilizada para facilitar a interação humano computador, baseada em ações previamente já conhecidas pelos usuários, como, por exemplo, voz e gestos. Nas interfaces CLI e GUI, os usuários precisam de um conhecimento prévio para saber o que digitar e/ou onde clicar para fazer determinada atividade. Na NUI, a interação é natural, pois utiliza a experiência pessoal dos usuários como ferramenta de comunicação, tornando possível escrever um texto apenas com a voz, ou então interagir com o dispositivo tocando na tela, recurso disponível na grande maioria dos celulares hoje em dia.

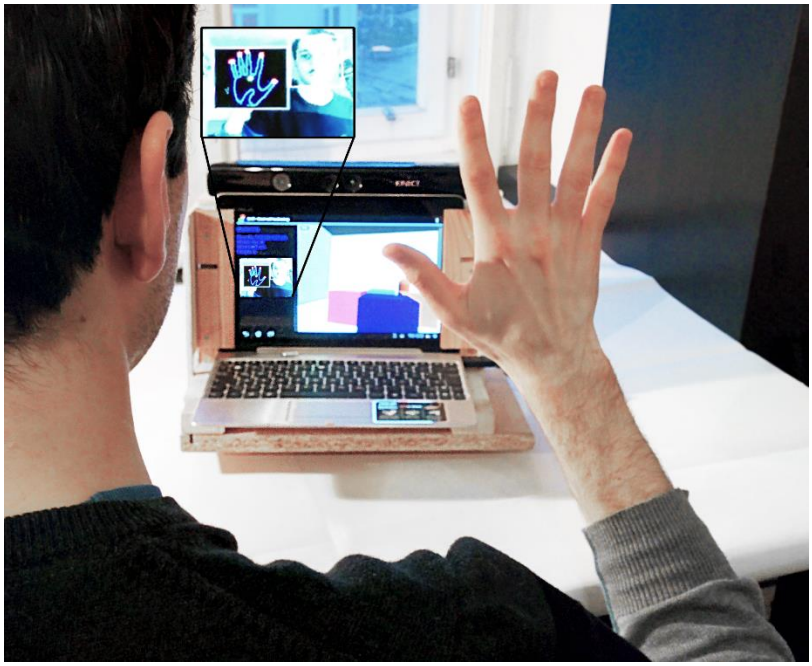
Segundo Weiyuan (2011), a NUI possui as seguintes características:

- Design centrado no usuário: provê as necessidades de mudança da interface para diferentes usuários;
- Multi-canal: visa capturar vários canais de entrada do usuário para a realização da interação com o computador. São considerados canais de entrada do usuário: mãos, boca, olhos, cabeça, pés e o corpo como um todo;
- Inexato: as ações e pensamentos dos usuários não são precisos, de modo que, o computador deve entender a requisição das pessoas;
- Largura de banda alta: devido à grande quantidade de dados gerados pelos dipositivos de interface utilizados, a NUI deve prover suporte a alta taxa de dados;
- Interação baseada por voz: considerada o meio mais conveniente, eficiente e natural de compartilhamento de informações. Em interfaces NUI possui duas abordagens: reconhecimento vocal e tecnologia de compreensão;
- Interação baseada por imagens: os humanos utilizam como principal sentido a visão. Para interfaces NUI, as imagens podem ser abordadas em 3 diferentes níveis: processamento de imagens, reconhecimento de imagens e percepção de imagens; e
- Interação baseada no comportamento: responsável por reconhecer a linguagem corporal do usuário, isto é, movimentos que expressem algum significado

Figura 3 - Tela sensível ao toque

Fonte: AGNI, Edu, 2015

Figura 4 - Reconhecimento de gestos



Fonte: VENDITTI, 2015

2.2 – Realidade Virtual

O termo Realidade Virtual foi criado no início dos anos 80, por Jaron Lanier, fundador da VPL Research Inc., para diferenciar as simulações tradicionais feitas por computador de simulações envolvendo múltiplos 5 usuários em um ambiente compartilhado (ARAÚJO, 1996). A Realidade Virtual é utilizada para descrever um ambiente tridimensional gerado por um computador, que pode ser explorado por uma pessoa. Essa pessoa se torna parte desse mundo virtual, e é capaz de manipular objetos ou executar uma série de ações.

O usuário pode ser inserido no mundo virtual através de dispositivos que transmitam sentidos como visão, audição, tato e olfato, transmitindo, em tempo real, a sensação de estar em um mundo paralelo ao mundo real.

A RV vem sendo cada vez mais utilizada, principalmente em jogos, com o objetivo de transportar os jogadores para dentro do jogo, geralmente tomando a forma de um personagem, fazendo com que a experiência seja mais próxima da realidade, comparada à experiência utilizando *joysticks*.

Na figura abaixo, um garoto está inserido em um ambiente virtual através de um óculos com reprodução de imagens e sensores de movimentos, e um fone de ouvido com reprodução de sons.

Figura 5 - Óculos de Realidade Virtual

Fonte: https://www.saybrook.edu

Também existem dispositivos mais completos que, além de óculos e fones de ouvido, oferecem objetos que simulam armas e esteiras para simular caminhada e corrida.

Figura 6 - Dispositivo de Realidade Virtual

Fonte: http://news.discovery.com/

2.3 – Realidade Aumentada

As bases da realidade aumentada surgiram na década de 1960, com o pesquisador norte americano Ivan Sutherland, que prestou duas contribuições principais: a) escreveu um artigo, vislumbrando a evolução da realidade virtual e seus reflexos no mundo real

(SUTHERLAND, 1965); b) desenvolveu um capacete de visão ótica direta rastreado para visualização de objetos 3D no ambiente real (SUTHERLAND, 1968).

No entanto, o primeiro projeto de realidade aumentada foi desenvolvido em 1980 pela Força Aérea Americana, sendo um simulador de *cockpit* de avião, misturando elementos virtuais com o ambiente físico do usuário.

Diferentemente da realidade virtual, que transporta o usuário para dentro de um ambiente virtual, a realidade aumentada mantém o usuário em seu ambiente físico e transporta o ambiente virtual para o espaço do usuário, por meio de dispositivos tecnológicos.

Milgran (1994) definiu realidade aumentada como a mistura de mundos reais e virtuais em algum ponto da realidade/virtualidade contínua, que conecta ambientes completamente reais a ambientes completamente virtuais.

A realidade aumentada mantém o senso de presença do usuário no mundo real através de recursos tecnológicos invisíveis para deixá-lo livre em seu ambiente. Recursos como sensores de direção, movimento e câmera são muito usados em aplicações de realidade aumentada. A figura abaixo demonstra um exemplo de realidade aumentada, que possibilita visualizar móveis em uma sala através de um *tablet.*

Figura 7 - Aplicativo de Realidade Aumentada

Fonte: http://syrusgold.com/

2.4 - Aplicações Existentes de Realidade Aumentada

Com o constante avanço da tecnologia, desenvolver aplicações de realidade aumentada está cada vez mais fácil e acessivo. Com isso, diversos projetos surgiram nos últimos anos, com diferentes propostas e objetivos.

2.4.1 - Second Surface

Um exemplo de aplicação de realidade aumentada é o projeto *Second Surface*, desenvolvido pela equipe do MIT Media Lab. Trata-se de uma aplicação que permite que os usuários façam desenhos tridimensionais, textos e fotos relativos a tais objetos e compartilhar essa expressão com qualquer outra pessoa que utiliza o mesmo software no mesmo local (MIT Media Lab).

Figura 8 - Second Surface

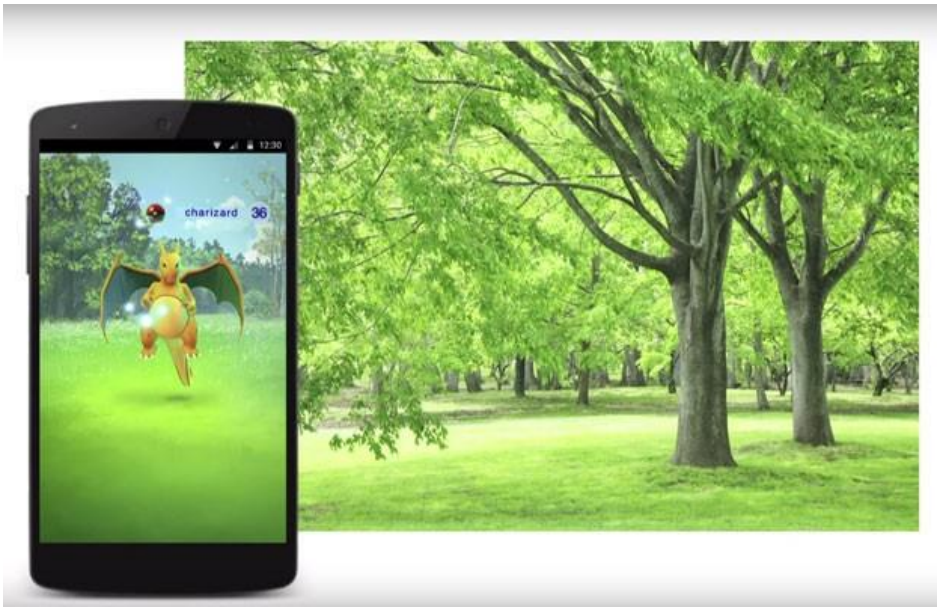
Fonte: VALENTIN HEUN (2012)

O Second Surface é um projeto fechado, não disponibilizado para a sociedade. Suas funcionalidades foram demonstradas apenas em eventos de tecnologia e através de vídeos criados pelo MIT Media Lab disponibilizados na internet.

2.4.2 – Pokemon Go

Um Jogo desenvolvido pela empresa Nintendo, que utiliza dados de localização para introduzir pokemons (personagens do jogo) no mundo real, e representá-los através de realidade aumentada. Assim, o jogador pode capturar, trocar e usar em batalhas com amigos. O lançamento do jogo está previsto para 2016, provavelmente no Brasil e no Mundo.

Figura 9 - Pokemon Go



Fonte: HEIDI HOOPES (2015)

2.4.3 – Coral Visualizer

O Coral Visualizer é um aplicativo fornecido pela empresa de tintas Coral, que permite pintar paredes utilizando realidade aumentada, assim ajudando o usuário a escolher a melhor cor de parede para cada ambiente. O aplicativo detecta ambientes e define bordas, superfícies e alterações no contorno, permitindo selecionar a área de uma imagem que será pintada. Também consegue detectar a diferença entre parede, móveis e demais objetos para colorir apenas a área da parede. Depois de escolher o local, ele seleciona automaticamente a parede e a preenche conforme o aparelho é movimentado, mostrando diferentes partes do ambiente. O Coral Visualizer está disponível para sistemas Android e IOS.

Figura 10 - Coral Visualizer

Fonte: http://lancou.com.br

3 - DESENVOLVIMENTO

Este capítulo aborda as etapas do desenvolvimento da aplicação Magic Surface, além de apresentar alguns trechos de código para melhor explicação das funcionalidades.

3.1 – Interface

A interface do Magic Surface utiliza o conceito *"Mobile Only*", que significa ser desenvolvida para uso exclusivo em dispositivos móveis. As páginas web que utilizam esse conceito ficam com aparência de aplicativos nativos, tornando a usabilidade mais intuitiva e confortável.

Para aplicar o conceito de *Mobile Only*, foi utilizado o Mobile Angular UI, um framework *front-end* baseado no Twitter Bootstrap para criar páginas web voltadas para dispositivos móveis.

3.1.1 – Utilizando o Mobile Angular UI

Para utilizar o Mobile Angular UI, é necessário importar o arquivo CSS "mobile-angular-ui-base.css" no cabeçalho do HTML através da seguinte *tag*:

```
<link rel="stylesheet" href="css/mobile-angular-ui-base.css">
```

3.1.1.1 – Barra de navegação

A barra de navegação, é uma barra localizada no topo da página que mostra o botão para abrir o menu de opções e o título da página. O fragmento de código abaixo mostra o código HTML que monta a barra de navegação utilizando classes do Mobile Angular UI.

Fragmento de Código 1 - Implementação da barra de navegação

```
1  <div class="navbar navbar-app navbar-absolute-top">
2      <div class="btn-group pull-left">
3          <a href="#menu-toggle" class="btn btn-default" id="menu-toggle"><i class="fa fa-bars"></i></a>
4      </div>
5      <div class="navbar-brand" ui-yield-to="title">
6          MAGIC SURFACE
7      </div>
8  </div>
```

Na linha 1, é definido o estilo da barra de navegação, através das classes `navbar` (define posição e tamanho da barra), `navbar-app` (define a cor de fonte e cor de fundo da barra) e `navbar-absolute-top` (define que a barra ficará no topo da tela). Na linha 2, é incluído o botão para abrir o menu de opções dentro da barra de navegação. São utilizadas as classes `btn-group` e `pull-left` para definir tamanho e posição do botão. Dentro da *tag*

do botão, na linha 3, está o link para a ação de abrir o menu, as classes `btn` e `btn-default` para dar a aparência de um botão e o ícone de barras através das classes `fa` e `fa-bars`.

Na linha 5, utilizando a classe `navbar-brand` (que define tamanho, posição e margem do componente), é definido o título da barra de navegação.

3.1.1.2 – Corpo da página

Corpo da página é onde fica o conteúdo principal, e utiliza a maior parte da tela. Para deixar o corpo da página nos padrões de telas de dispositivos móveis, é utilizada a classe CSS `page-content-wrapper` em uma `div` HTML, onde todas as `div's` filhas serão parte do corpo da página.

3.2 – AngularJS

O AngularJS é um framework estrutural para desenvolvimento de aplicativos Web dinâmicos. Ele permite a utilização do HTML como um template, possibilitando que a sintaxe HTML seja estendida para representar de forma mais clara e sucinta os componentes da aplicação. O AngularJS é um arquivo JavaScript, não necessitando procedimentos de instalação. (ANGULARJS).

Para utilizar o angularjs na página, primeiramente é necessário importar o arquivo angular.min.js no código HTML. Em seguida deve-se definir uma aplicação e um controlador angular, para isso, segue o código javascript:

Fragmento de Código 2 - Criando uma aplicação e um controlador Angular

```
1  angular.module('MagicApp',['ui.bootstrap','magicSurface']);
2
3  angular.module('MagicApp').controller('MainCtrl', function($scope, MagicSurface, LayerApi, FileApi) {
4
5  }
```

No fragmento de código acima, na linha 1, é definida a aplicação angular, denominada "*MagicApp*", e em seguida são definidas as dependências "ui.bootstrap" e "magicSurface", responsáveis respectivamente por oferecer efeitos visuais dinâmicos e funções para comunicação com o servidor. Na linha 3, é definido o controlador angular, denominado "*MainCtrl*", e é chamada uma função, que recebe alguns parâmetros para manipulação dos dados, e contém toda a lógica da página.

No código HTML, é necessário declarar a aplicação "*MagicApp*" e o controlador "*MainCtrl*". Para isso, são inseridos dois parâmetros na *tag* `body`, "ng-app" e ng-controller", da seguinte forma:

```
<body ng-app='MagicApp' ng-controller="MainCtrl">
```

Desta forma, todos componentes dentro da *tag* `body` serão controlados pelo angular.

3.3 – API

Este trabalho de graduação aborda o desenvolvimento de uma interface para uma aplicação de realidade aumentada. Para manipulação dos dados da aplicação, é utilizada uma API desenvolvida em outro Trabalho de Graduação da FATEC de São José dos Campos. Esta API fornece funções javascript para salvar, editar e listar dados em um servidor. A documentação desta API está disponível em http://magicsurfacebr.appspot.com.

Vale ressaltar que, juntos, os dois trabalhos de graduação formam uma aplicação final.

3.4 – Mapa e Geolocalização

O aplicativo Magic Surface utiliza a localização do usuário para representa-lo no mapa, criar e visualizar *Layers*. O fragmento abaixo mostra a requisição da localização do usuário através da API Geolocation e a criação do mapa através da API do Google Maps.

Fragmento de Código 3 - Consultando localização e atualizando mapa

```
1   var userPosition;
2   var map;
3   var marker;
4
5   navigator.geolocation.getCurrentPosition(function(position){
6       userPosition = new google.maps.LatLng(position.coords.latitude, position.coords.longitude);
7       var mapOptions = {
8           center: userPosition,
9           disableDefaultUI: true,
10          zoom: 17
11      }
12      map = new google.maps.Map(document.getElementById('map'), mapOptions);
13      marker = new google.maps.Marker({
14          position: userPosition,
15          map: map
16      });
17  });
```

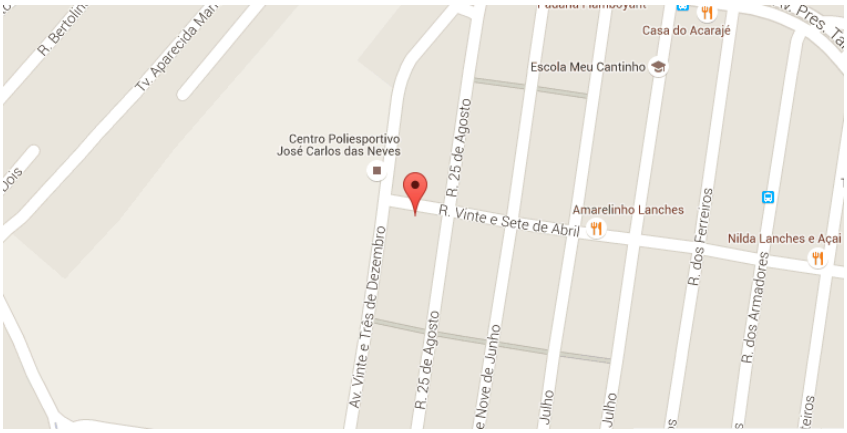
Nas linhas 1, 2 e 3, são inicializadas as variáveis `userPosition`, `map` e `marker`. Na linha 5 é feita a requisição da localização do usuário através do método `getCurrentPosition` que recebe uma função com o parâmetro `position`. Na linha 6, a variável `userPosition` recebe um objeto `google.maps.LatLng` com a localização obtida através do parâmetro `position` representada como latitude e longitude. Na linha 12, o mapa é criado através do objeto `google.maps.Map` e atribuído à variável `map` com as opções de mapa definidas nas linhas 7 à 11. Um marcador é criado na linha 13 através do objeto `google.maps.Marker` e atribuído à variável `marker`.

Para mostrar o mapa na tela, é inserida uma *div* com o id "map" no HTML conforme demonstrado abaixo:

```
<div id="map"></div>
```

A figura a seguir demonstra o resultado obtido:

Figura 11 - Mostrando a localização no mapa



3.5 – Layer

Layer é um raio no espaço terrestre com tamanho definido pelo usuário, onde são registrados fotos e vídeos que podem ser visualizados por qualquer pessoa dentro do *layer*. Para manipular as informações dos *layers* no servidor, é utilizada a API MagicSurface LayerAPI, que fornece métodos para salvar, listar e apagar *layers*.

As subseções a seguir abordam como é feita a criação e listagem de *layers*.

3.5.1 – Criação

Um *layer* é composto por Nome, Raio (em metros), Latitude e Longitude. O fragmento abaixo mostra o formulário HTML para criação de um *Layer*. Os dados são inseridos em um `form` através do `ng-model` que é refletido diretamente no javascript. Para

submeter os dados, é chamada a função `submitLayer()` no botão Criar através do `ng-click`.

Fragmento de Código 4 - Formulário html para criação de um layer

```
1  <div class="form col-lg-12">
2      <div style="color:red">
3          {{mensagem}}
4      </div>
5      <div class="form-group has-success has-feedback">
6          <input type="text" ng-model="form.name" class="form-control ng-pristine ng-valid">
7      </div>
8      <div class="form-group has-success has-feedback">
9          <input type="number" ng-model="form.radius" class="form-control ng-pristine ng-valid">
10     </div>
11     <div class="form-group">
12         <p> Latitude: {{latitude}} </p>
13         <p> Longitude: {{longitude}} </p>
14     </div>
15     <div class="form-group">
16       <button type="submit" ng-click="submitLayer()" class="btn btn-primary btn-block">Criar</button>
17     </div>
18 </div>
```

No fragmento abaixo, o *layer* é enviado para o servidor utilizando a API Magic Surface.

Fragmento de Código 5 - Implementação da função submitLayer

```
1  $scope.submitLayer = function() {
2
3      $scope.form["latitude"] = userPosition.lat();
4      $scope.form["longitude"] = userPosition.lng();
5
6      var promise = LayerApi.save($scope.form);
7
8      $scope.ajaxload = true;
9
10     promise.success(function(result){
11         $scope.descricao = result.name;
12         $scope.selected_layer = result;
13         $scope.mostrarLayer();
14         $scope.ajaxload = false;
15     });
16
17     promise.error(function(result){
18         $scope.mensagem = "Erro ao enviar formulário. " ;
19         $scope.ajaxload = false;
20         console.log(result);
21     });
22 };
```

Nas linhas 3 e 4, a latitude e longitude é incluída no `form` junto às informações obtidas no formulário html. Na linha 6, é chamada a função `LayerApi.save` e passado o `form` como parâmetro. O LayerApi fornece duas respostas assíncronas, *success* e *error*. Na linha 10 é chamada a função `success` e passada uma função como parâmetro que retorna um resultado com o *layer* salvo. Na linha 11, o escopo `descricao` recebe o nome do *layer*.

Na linha 12, o escopo `selected_layer` recebe o resultado da função de sucesso com o *layer* salvo. Na linha 13, é chamada a função `mostrarLayer()` que mostra o *layer* na tela.

3.5.2 – Listagem

Os *layers* são exibidos no mapa de acordo com sua localização e tamanho do raio. A listagem é feita através da função `getLayers()` conforme fragmento abaixo:

Fragmento de Código 6 - Implementação da função getLayers

```
1   var layers = new Array();
2
3   function getLayers() {
4       var promise = LayerApi.list();
5       $scope.ajaxload = true;
6       promise.success(function(result){
7           layers = result.layers;
8           criarLayers();
9           $scope.ajaxload = false;
10      });
11      promise.error(function(result){
12          alert("Erro ao buscar layers.");
13          $scope.ajaxload = false;
14      });
15  }
```

Dentro da função `getLayers`, na linha 4 é chamada a função `LayerApi.list`, que possui duas chamadas de resposta, `success` e `error`. Na chamada `success`, na linha 7, o resultado obtido é inserido na variável `layers`. Em seguida é chamada a função `criarLayers`, para mostrar os layers na tela. Na chamada `error`, uma mensagem de erro é mostrada na tela.

Fragmento de Código 7 - Implementação da função criarLayers

```
var criarLayers = function(){

    var layerOptions = {
        strokeColor: '#1000FF',
        strokeOpacity: 0.8,
        strokeWeight: 2,
        fillColor: '#1000FF',
        fillOpacity: 0.35,
        map: map,
        center: null,
        radius: null
    };

    for(var i=0; i<layers.length; ++i){
        layerOptions.center = new google.maps.LatLng(layers[i].latitude,layers[i].longitude);
        layerOptions.radius = layers[i].radius;
        var circle = new google.maps.Circle(layerOptions);
        var infowindowLayer = new google.maps.InfoWindow({
            map: map,
            position: layerOptions.center,
            content: layers[i].name
        });
    }
}
```

A exibição de layers no mapa é feita através de círculos fornecidos pela API do Google Maps. O fragmento de código acima, mostra a função criarLayers, que percorre a lista de layers, e cria um circle (círculo) para cada layer, de acordo com sua posição e raio.

3.6 – Análise de Posição

A análise de posição é uma verificação feita a cada um segundo, que consulta a posição do usuário, atualiza a posição no mapa e compara com a posição dos *layers* existentes. Caso a análise retorne que o usuário está dentro do raio de um *layer*, o nome do *layer* é mostrado na tela, tornando possível acessá-lo, para enviar ou visualizar arquivos.

Para fazer a análise de tempos em tempos, é utilizado um *timer*, onde é possível definir um tempo de intervalo e uma função para ser executada a cada repetição:

```
var timer = $interval(autoUpdate,1000);
```

No código acima, a variável `timer` recebe um objeto de intervalo, que executará a função `autoUpdate` a cada 1000 milissegundos.

Fragmento de Código 8 - Implementação da função autoUpdate

```
1  function autoUpdate() {
2      navigator.geolocation.getCurrentPosition(displayPosition, displayError);
3      function displayPosition(position) {
4          userPosition = new google.maps.LatLng(position.coords.latitude, position.coords.longitude);
5          marker.setPosition(userPosition);
6          map.setCenter(userPosition);
7          for(var i=0; i<layers.length; ++i){
8              layerPosition = new google.maps.LatLng(layers[i].latitude, layers[i].longitude);
9              if(calcDistance(userPosition, layerPosition) < layers[i].radius) {
10                 $scope.descricao = layers[i].name;
11                 $scope.selected_layer = layers[i];
12                 $scope.btnAbrir = true;
13                 $scope.btnCriar = false;
14             }
15         }
16     }
17     function displayError(error) {
18         var errors = {
19             1: 'Permission denied',
20             2: 'Position unavailable',
21             3: 'Request timeout'
22         };
23         alert("Error: " + errors[error.code]);
24     }
25 }
```

Na função `autoUpdate`, na linha 2, é feita a requisição de posição do usuário, chamando as funções de sucesso e de erro, respectivamente, `displayPosition` e `displayError`. Dentro da função `displayPosition`, na linha 4, a variável `userPosition` recebe a posição do usuário retornada como parâmetro da função. Nas linhas 5 e 6, o marcador e o centro do mapa são atualizados com a nova posição. A partir da linha 7, uma estrutura de repetição percorre a lista de *layers*, e compara com a posição do usuário, através da função `calcDistance`. Caso a comparação retorne que a distância entre o usuário e o *layer* é menor que o raio, a tela é atualizada com as informações do *layer* e funções para visualizá-lo. Na função de erro, a partir da linha 17, um alerta é mostrado na tela com o erro recebido como parâmetro da função.

A figura abaixo, mostra a implementação da função `calcDistance`.

Fragmento de Código 9 - Implementação da função calcDistance

```
1  function calcDistance(p1, p2){
2      return (google.maps.geometry.spherical.computeDistanceBetween(p1, p2)).toFixed(2);
3  }
```

A função `calcDistance` recebe dois valores como parâmetro, a posição do usuário e a posição do *layer*, e utiliza a função `computeDistanceBetween`, fornecida pela API do Google Maps, para retornar a distância entre as duas posições.

3.7 – Arquivos

Os arquivos são fotos e vídeos registrados pelo usuário que podem ser salvos em um *layer*. Um arquivo é formado pela imagem ou vídeo, e as coordenadas da posição do

dispositivo, composta por três valores (x, y, z), que são analisados para saber em qual direção o usuário está apontando o dispositivo no momento da captura. Baseado nestas coordenadas, o arquivo é mostrado na tela conforme a movimentação do dispositivo.

Para manipular as informações dos arquivos no servidor, é utilizada a API MagicSurface FileAPI, que fornece métodos para salvar, listar e apagar arquivos.

As subseções a seguir abordam como é feito o envio, listagem e visualização dos arquivos.

3.7.1 – Envio

O envio de arquivos é feito através da função `submitFile`. O fragmento de código abaixo mostra o código HTML para envio do arquivo.

Fragmento de Código 10 - Implementação do HTML para envio de arquivos

```
1  <div class="form-group has-success has-feedback text-center">
2      <span class="btn btn-primary btn-file">
3          Adicionar Foto / Vídeo ... <input type="file" name="file" ng-model="file"
4          onchange="angular.element(this).scope().submitFile(this.files)"/>
5      </span>
6  </div>
```

Para enviar o arquivo, é utilizado um `input`, na linha 3, do tipo “file”, nome “file” e modelo angular “file”. Ao selecionar um arquivo, é acionada a função “`onchange`” que chama a função “`submitFile`” passando o arquivo selecionado como parâmetro.

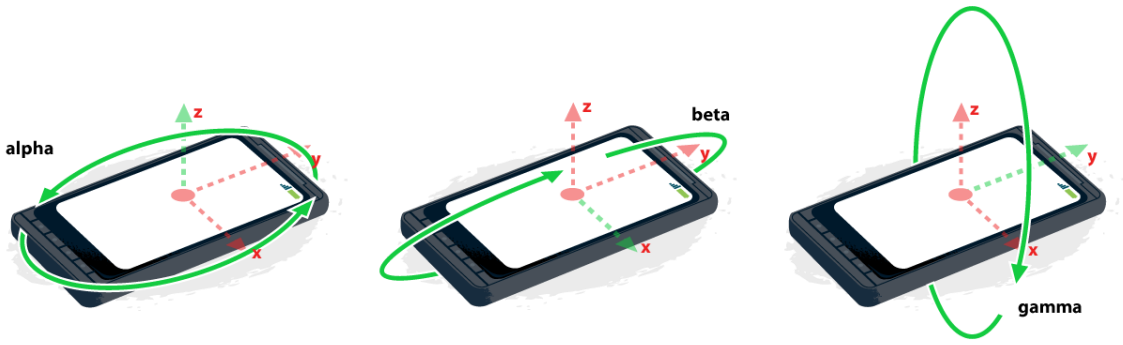
O fragmento abaixo mostra a implementação em javascript para envio dos arquivos para o servidor.

Fragmento de Código 11 - Implementação da função submitFile

```
1  var x=0, y=0, z=0;
2
3  window.addEventListener('deviceorientation', function(event) {
4      x = event.alpha.toFixed(0);
5      y = event.beta.toFixed(0);
6      z = event.gamma.toFixed(0);
7  });
8
9  $scope.submitFile = function(files){
10     var params = {
11         layerId: $scope.selected_layer.id,
12         angle_x: x,
13         angle_y: y,
14         angle_z: z
15     }
16     var _file = files[0];
17     var promise = FileApi.save(_file, params);
18     $scope.ajaxload = true;
19     promise.success(function(result){
20         if(result.kind == "image"){
21             $scope.imgs.push(result);
22             console.log($scope.imgs);
23         }
24         else if(result.kind == "video"){
25             $scope.videos.push(result);
26             console.log($scope.videos);
27         }
28         $scope.ajaxload = false;
29     });
30     promise.error(function(result){
31         $scope.ajaxload = false;
32         console.log(result);
33     });
34 };
```

Na linha 1, são declaradas as variáveis `x`, `y` e `z`, para guardar os valores obtidos no evento `deviceorientation`, definido na linha 3. Este evento do HTML5, é responsável por consultar o giroscópio do dispositivo, e retornar três valores: *alpha*, *beta* e *gamma*. A figura abaixo ilustra como funciona a orientação do dispositivo baseada nestes valores, onde *alpha* é a rotação no eixo X, *beta* é a rotação no eixo Y, e *gamma* é a rotação no eixo Z.

Figura 12 - Orientação do dispositivo



Fonte: JEREMIE, 2014

A partir da linha 9, é definida a função `submitFile`, que chama a função `FileApi.save`, que recebe como parâmetro, o arquivo e um objeto de parâmetros, contendo o *id* do *layer* onde o arquivo será inserido, e os três ângulos obtidos no evento `deviceorientation`, denominados `angle_x`, `angle_y` e `angle_z`. Na linha 19, é chamada a função de sucesso enviada pelo servidor, recebendo o arquivo enviado através do parâmetro `result`. A partir da linha 19, é feita a verificação do arquivo, para definir se é uma imagem ou vídeo, e inseri-los em suas respectivas listas. Na linha 30, é chamada a função de erro, também recebendo o parâmetro `result`, contendo a mensagem de erro, que é mostrada na tela.

3.7.2 – Listagem

A listagem dos arquivos é feita através da função `getFiles`. A figura abaixo mostra a implementação desta função:

Fragmento de Código 12 - Implementação da função getFiles

```
1   $scope.imgs = [];
2   $scope.videos = [];
3
4   var getFiles = function(layer){
5       $scope.imgs = [];
6       $scope.videos = [];
7       var params = {
8           layerId: layer.id,
9           filters: {
10              files: 'all',
11              deleted: false
12          }
13      };
14      var promise = FileApi.list(params);
15      $scope.ajaxload = true;
16      promise.success(function(result){
17          var files = result.files;
18          for(var i=0; i<files.length; i++){
19              if(files[i].kind == "image"){
20                  $scope.imgs.push(files[i]);
21              }
22              else if(files[i].kind == "video"){
23                  $scope.videos.push(files[i]);
24              }
25          }
26          $scope.ajaxload = false;
27      });
28      promise.error(function(result){
29          alert("Erro ao buscar imagens");
30          $scope.ajaxload = false;
31      });
32  }
```

Dentro da função, é criado o objeto `params`, que contém um `layerId` e um `filters` para definir que a busca deve retornar todos arquivos que não foram deletados,

através dos valores: “`files: 'all'`” e “`deleted: false`”. Logo em seguida, na linha 14, é chamada a função FileApi.list, passando o objeto params como parâmetro. Na função de sucesso, a partir da linha 16, a variável files recebe os arquivos obtidos pelo resultado da chamada. Em seguida é feita uma verificação na lista de arquivos através de uma estrutura de repetição para atribuir as imagens na lista de imagens e os vídeos na lista de vídeos. A partir da linha 28, é declarada a função de erro, que mostra uma mensagem de erro na tela.

3.7.3 – Visualização

A visualização dos arquivos é feita de duas formas: estaticamente e em modo realidade aumentada. A visualização estática apresenta as imagens e vídeos na tela, de acordo com o *layer* selecionado. A visualização em modo RA, permite visualizar imagens no mundo real através da câmera, baseadas em suas coordenadas de posição, como, por exemplo, caso a imagem seja registrada apontando o dispositivo para o leste, posicionado verticalmente, somente será possível visualizá-la ao apontar o dispositivo para a mesma direção.

3.7.3.1 – Visualização estática

Para visualizar os arquivos na tela, é utilizado o código HTML abaixo:

Fragmento de Código 13 - HTML para visualização dos arquivos

```
1  <div ng-repeat="i in imgs" active="i.active" class="file text-center">
2      <button class="btn btn-default" ng-click="mostrarCamera(i)">
3          Visualizar em modo Realidade Aumentada
4      </button>
5      <img ng-src="{{i.link}}">
6  </div>
7
8  <player ng-repeat="video in videos" video='video' />
```

O AngularJS fornece algumas diretivas para integração do HTML com o JavaScript, uma delas é o `ng-repeat`, que percorre uma lista javascript repetindo a *tag* de acordo com o número de elementos da lista. No fragmento de código acima, a diretiva ng-repeat é utilizada para percorrer as listas de imagens e vídeos e exibi-los na tela. Nas linhas de 1 a 6, as imagens são renderizadas na tela juntamente com um botão para abrir a exibição em modo realidade aumentada. Na linha 8, é chamada a diretiva player para listagem dos vídeos. Abaixo, segue o código da diretiva player, que fornece algumas funcionalidades básicas para reprodução de vídeos, como funções de *Play* e *Pause*.

Fragmento de Código 14 - Diretiva Player

```
1  angular.module('MagicApp').directive('player', ['$sce', function ($sce) {
2      'use strict';
3      return {
4          restrict: 'E',
5          scope: {
6              video: '='
7          },
8          link: function (scope, element, attrs) {
9              var video = element.find('video');
10             element.addClass('player');
11             scope.playing = false;
12             scope.trustSrc = function(src) {
13                 return $sce.trustAsResourceUrl(src);
14             }
15
16             video.on('timeupdate', function (e) {
17                 scope.$apply(function () {
18                     scope.percent = (video[0].currentTime / video[0].duration) * 100;
19                 });
20             });
21
22             scope.frame = function (num) {
23                 if (video[0].readyState !== 0) {
24                     video[0].currentTime += num;
25                 }
26             };
27
28             scope.toggle = function () {
29                 if (video[0].paused === true) {
30                     video[0].play();
31                     scope.playing = true;
32                 } else {
33                     video[0].pause();
34                     scope.playing = false;
35                 }
36             };
37         },
38         template: '<a ng-click="toggle()"><video>' +
39             '<source ng-src="{{ trustSrc(video.link) }}" type="video/mp4" />' +
40             '</video>' +
41             '</a>'
42     };
43 }]);
```

Na linha 1, a diretiva player é definida como parte da aplicação `MagicApp`. Na linha 4, o atributo "`restrict: 'E'`", define que a diretiva será chamada no HTML como uma *tag*, exemplo: `<player>`. A partir da linha 8, é iniciada a função principal da diretiva. Na linha 9, o elemento `video` é selecionado e em seguida, uma classe CSS é aplicada para definir o estilo. Na linha 16, é chamada a função `video.on`, para iniciar a reprodução do vídeo. Na linha 22, é definida a função `frame`, que irá exibir o tempo de reprodução. Na linha 28, a função `toggle` é responsável por iniciar e pausar o vídeo. A partir da linha 38, é feita a montagem do código HTML que será substituído pela *tag* `<player>`.

3.7.3.2 – Visualização em Modo Realidade Aumentada

A visualização em modo RA, é baseada em duas camadas sobrepostas no código HTML. A primeira camada é o componente de vídeo, que representa a câmera do dispositivo. A segunda camada é o *canvas*, um componente HTML capaz de renderizar elementos gráficos. Desta forma, com as camadas sobrepostas, é possível transmitir a sensação de que os elementos estão na imagem da câmera. Para isso, primeiramente, são definidas as camadas no código HTML, conforme fragmento de código abaixo:

Fragmento de Código 15 - HTML para modo RA

```
1  <canvas id="myCanvas" width="350" height="480">
2      Your browser does not support the HTML5 canvas tag.
3  </canvas>
4
5  <div style="position:absolute;">
6      <video id="sourcevid" autoplay width="350">
7          Sorry, you're browser doesn't support video. Please try
8      </video>
9  </div>
```

Nas linhas de 1 a 3, é definido o *canvas*, com um id, e um tamanho fixo de 350 pixels de largura por 480 pixels de altura. Nas linhas de 5 a 9, é definido o componente de vídeo, com uma posição absoluta, para criar o efeito de sobreposição entre as camadas, um código identificador, e uma largura fixa, com o mesmo tamanho de largura do *canvas*.

Para capturar as imagens da câmera, é utilizada a API GetUserMedia, fornecida pelo HTML5. Esta API permite manipular recursos de mídia dos dispositivos, como, por exemplo, câmera e microfone. No código abaixo, é feita a captura da imagem através do método `navigator.getUserMedia`, e atribuída ao elemento de vídeo inserido no HTML.

Fragmento de Código 16 - Utilizando getUserMedia

```
1  if (navigator.getUserMedia) {
2      navigator.getUserMedia(function(stream) {
3          var video = document.getElementById('sourcevid');
4          video.src = window.URL.createObjectURL(stream);
5          video.onloadedmetadata = function(e) {
6              video.play();
7          };
8      }, function(err) {
9          alert("The following error occured: " + err.name);
10     });
11 } else {
12     alert("getUserMedia not supported");
13 }
14 video = document.getElementById('sourcevid');
```

Após definido o vídeo, é feita a definição do *canvas*, começando pelo método `abrirModoRA`. Este método é acionado ao selecionar uma das imagens em modo estático.

Fragmento de Código 17 - Implementação da função abrirModoRA

```
1  $scope.abrirModoRA = function(image) {
2      $scope.raVisible = true
3      $scope.mapaVisible = false;
4      $scope.formVisible = false;
5      $scope.conteudoLayer = false;
6
7      var c=document.getElementById("myCanvas");
8      var ctx=c.getContext("2d");
9      var img = new Image();
10     img.src = image.link;
11     img.onload = function () {
12         var alpha=0, beta=0, gamma=0;
13         window.addEventListener('deviceorientation', function(event) {
14             alpha = event.alpha.toFixed(0);
15             beta = event.beta.toFixed(0);
16             gamma = event.gamma.toFixed(0);
17             if(verifyOrientation(alpha, beta, gamma, image.angle_x, image.angle_y, image.angle_z)) {
18                 ctx.drawImage(img, 70, 50, 200, 300);
19             }
20             else {
21                 ctx.clearRect(0, 0, c.width, c.height);
22             }
23         });
24     }
25 }
```

No código acima, das linhas 2 a 5, são alteradas as variáveis que gerenciam os componentes na tela, para esconder a tela de visualização estática e mostrar a tela de visualização em modo RA. Nas linhas 7 e 8, o elemento *canvas* é selecionado e atribuído a um contexto 2d (duas dimensões), representado pela variável `ctx`. Nas linhas 9 e 10, é criado um objeto de imagem, com o caminho da imagem selecionada pelo usuário, recebida como parâmetro do método. Nas linhas 13 a 23, é chamado o evento de orientação do dispositivo, que será acionado a cada movimentação que o usuário fizer, e baseado nos valores de alpha, beta e gamma, é feita a verificação se estas coordenadas se aproximam das coordenadas da imagem, através da função `verifyOrientation`. Caso a função retornar verdadeiro, a imagem é renderizada no *canvas*. Caso falso, o *canvas* é limpo e a imagem some.

Fragmento de Código 18 - Implementação da função verifyOrientation

```
1  function verifyOrientation(a, b, g, x, y, z) {
2      if(a > (x-15) && a < (x+15) && b > (y-15) && b < (y+15) && g > (z-15) && g < (z+15)) {
3          return true;
4      }
5      return false;
6  }
```

A função verifyOrientation recebe seis valores como parâmetro, sendo três coordenadas da posição atual e três coordenadas da imagem. A verificação é feita com uma margem de 15 graus, para melhor visualização da imagem, levando em conta a oscilação dos sensores de movimento e direção.

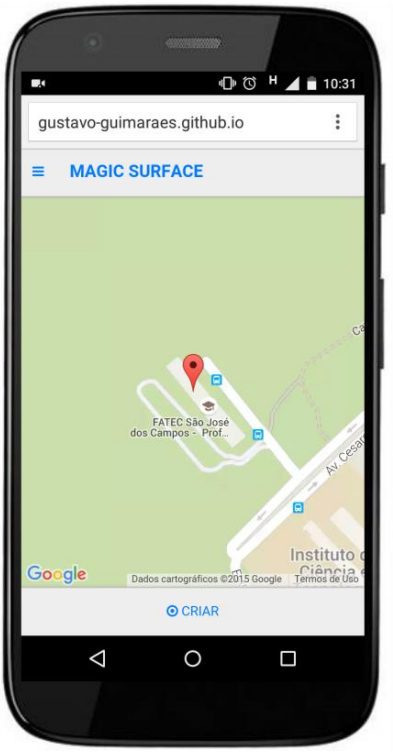
4 - RESULTADOS

Este capítulo aborda os resultados obtidos com a aplicação Magic Surface.

4.1 – Mapa

A tela inicial da aplicação é a tela do Mapa, nela é possível visualizar a posição do usuário em tempo real, bem como criar um novo *layer* a partir do botão “Criar”, localizado na parte inferior da tela.

Figura 13 - Tela de visualização do mapa



4.2 – Layer

A partir da tela do mapa, é possível criar e visualizar Layers. As subseções a seguir abordam como realizar tais procedimentos.

4.2.1 – Criação

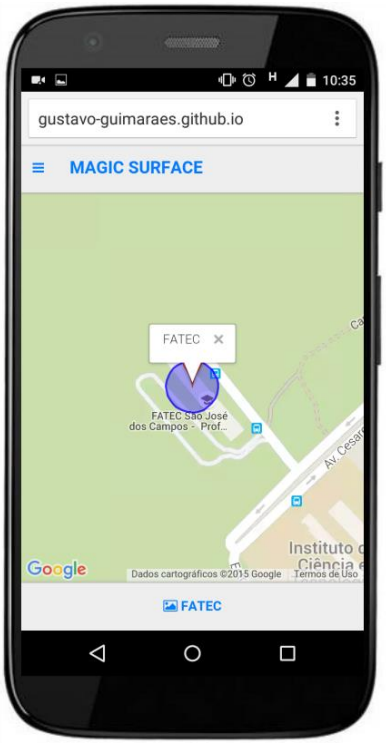
Ao clicar no botão “Criar”, é exibido o formulário para criação de Layers, conforme figura abaixo. O formulário contém os campos Nome e Raio para serem preenchidos, e a latitude e longitude da posição do usuário. Abaixo, estão os botões de “Criar” e “Cancelar”.

Figura 14 - Tela de criação de Layer

4.2.2 – Visualização

Após criar um Layer, ele é exibido no mapa através de um círculo com o tamanho especificado no Raio, e o Nome do Layer, conforme figura abaixo.

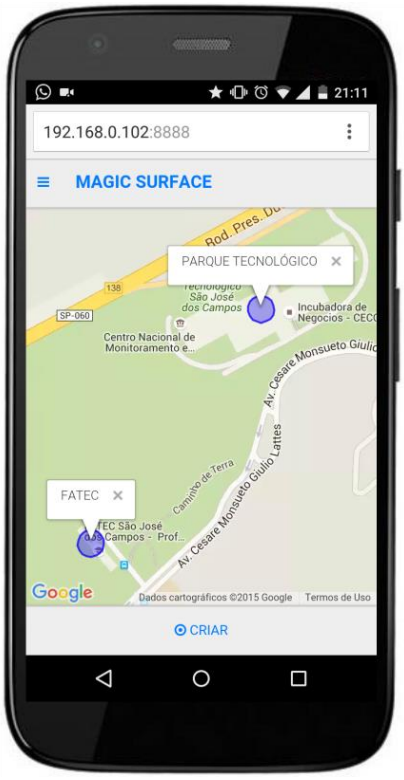
Figura 15 - Tela do mapa com layer



Na parte inferior da tela, é exibido um botão com o nome do *layer*, neste caso, “FATEC”. Isso significa que o usuário está posicionado geograficamente dentro do raio do *layer*, assim podendo enviar e visualizar os arquivos deste.

É possível visualizar mais *layers* ao expandir o mapa.

Figura 16 - Tela do mapa com múltiplos layers



4.3 – Análise de Posição

A análise de posição, feita de tempos em tempos, é responsável por atualizar a tela com a posição do usuário e verifica se o mesmo está dentro de um *layer* ou não. Estando fora do *layer*, o botão de Criar fica visível possibilitando criar novos *layers*. Caso esteja dentro de um *layer*, o botão é alterado com o nome do *layer*, para visualização do seu conteúdo. A figura abaixo simula a análise de posição de um usuário em movimento, onde o marcador vermelho representa o usuário e o círculo azul representa o *layer*.

Figura 17 - Análise de Posição

Observe que ao entrar no *layer*, o botão de criar é alterado para o botão com o nome do *layer*, neste caso “FATEC”.

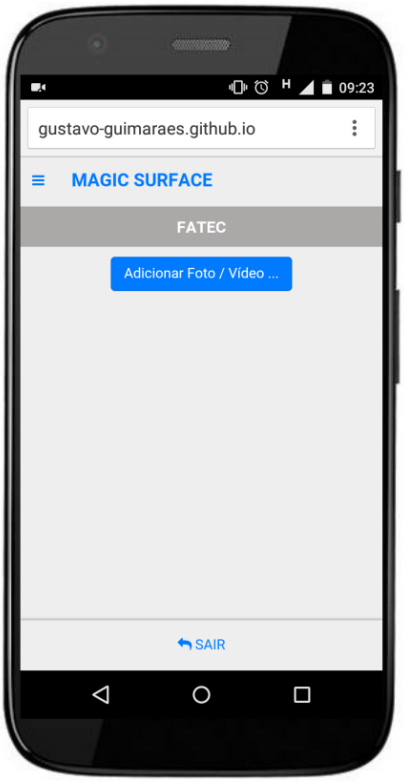
4.4 – Arquivos

As subseções a seguir abordam os resultados obtidos para envio e visualização de arquivos.

4.4.1 - Envio

Para enviar arquivos, basta clicar no botão com o nome do *layer*, localizado na parte inferior da tela. Ao clicar, será exibida a tela de informações do *layer*, conforme figura abaixo:

Figura 18 - Tela de informações do layer



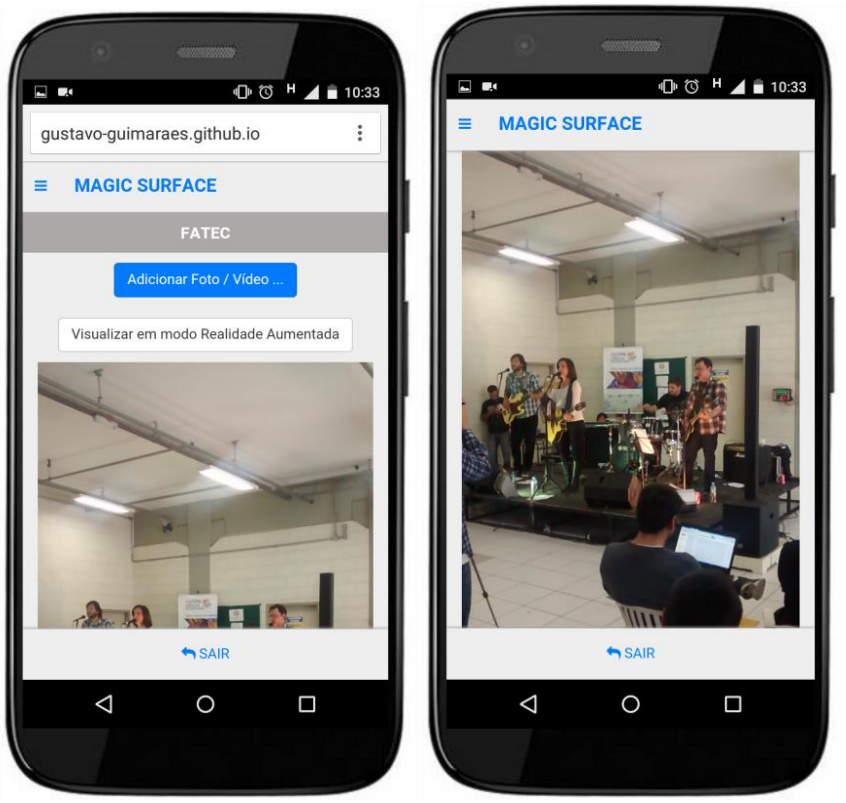
A tela de informações do *layer* possui, na parte superior, o título da página com o nome do *layer*, seguido do botão para enviar fotos ou vídeos, e, ao final, um botão para sair desta tela e voltar para a tela do mapa.

Ao clicar no botão “Adicionar Foto / Vídeo ...”, a câmera do dispositivo será acionada, permitindo registrar uma foto ou vídeo. Após terminar a captura, a imagem é enviada para o servidor junto com as informações de posição geográfica e coordenadas de rotação do dispositivo.

4.4.2 – Visualização

Ao enviar um arquivo, a tela de informações do *layer* é atualizada, mostrando a imagem ou o vídeo que foi enviado.

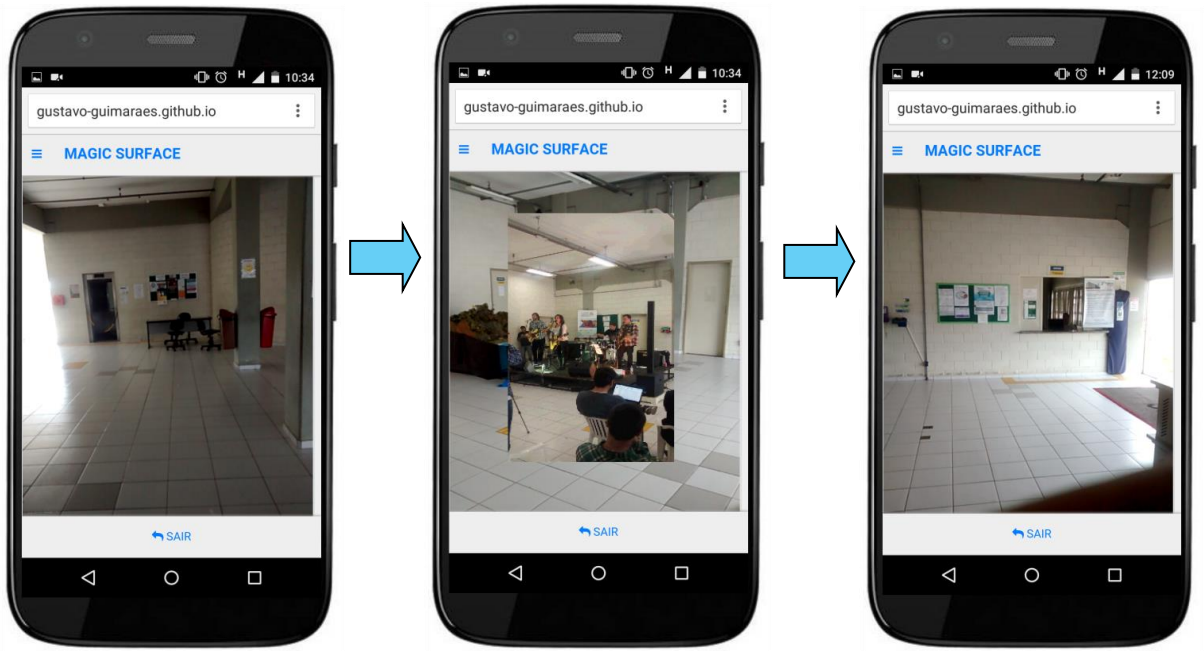
Figura 19 - Visualização estática de arquivos



A figura acima, mostra a visualização de arquivos em modo estático, onde as imagens e vídeos são apresentados em forma de lista, um em baixo do outro.

Para visualizar os arquivos em Modo Realidade Aumentada, o botão “Visualizar em modo Realidade Aumentada”, a cima de cada imagem, abrirá a câmera do dispositivo, com a imagem selecionada em sua posição de origem, sendo necessário apontar para o local onde a mesma foi registrada para visualizá-la. A figura abaixo mostra três estados de uma visualização em modo RA. O primeiro estado é apontando para o lado esquerdo em relação a posição do arquivo, onde não é exibido nada. O segundo, o smartphone está apontando para o local onde o arquivo foi registrado, assim visualizando o arquivo na câmera. O terceiro e último estado está apontando para o lado direito em relação a posição do arquivo, onde também não é possível visualizá-lo.

Figura 20 - Visualização em modo RA



A figura abaixo mostra o pátio da FATEC São José dos Campos, local onde a imagem acima foi registrada. Esta foto foi tirada dias depois da apresentação da banda de rock, que foi registrada na aplicação e agora pode ser visualizada a qualquer momento.

Figura 21 - Pátio da FATEC SJC

4.6 – Aplicação Final

A aplicação Magic Surface está disponível para uso na plataforma GitHub, no endereço: http://gustavo-guimaraes.github.io/MagicSurface/.

Figura 22 - Código QR para acessar a aplicação

O código fonte da aplicação está disponível também no GitHub, no endereço: https://github.com/gustavo-guimaraes/MagicSurface/.

Figura 23 - Código QR para código fonte da aplicação

5 - CONSIDERAÇÕES FINAIS

Este trabalho apresentou o desenvolvimento de uma interface web para uma aplicação de realidade aumentada, para ser utilizada em dispositivos móveis, independente do sistema operacional. Com esta aplicação, é possível registrar imagens e vídeos baseados na localização geográfica e posição do dispositivo, para serem visualizados através da câmera de um *smartphone* ou *tablet*. Na Seção 5.1 serão abordadas as contribuições obtidas ao longo do desenvolvimento deste trabalho, e na Seção 5.2 serão sugeridos alguns trabalhos futuros.

5.1 – Contribuições e Conclusões

As contribuições deste trabalho são:

- Implementação de uma aplicação de realidade aumentada;
- Manipulação de sensores em dispositivos móveis através de uma aplicação web;
- Uso de interfaces naturais;
- Estudo de novas formas de utilização da realidade aumentada;
- Disponibilização do código *open-source;*
- Disponibilização da aplicação para a sociedade.

A partir destas contribuições pode se concluir que:

- A aplicação desenvolvida oferece uma forma inovadora de registrar imagens e vídeos no mundo real através da realidade aumentada;
- Está disponível em qualquer lugar com acesso à internet;
- É possível utilizar em qualquer *smartphone* ou *tablet*, independente do sistema operacional;
- A realidade aumentada engaja os usuários a utilizarem a aplicação, por se tratar de uma interface interessante e divertida;

5.1.1 - Publicações

Primariamente, foi publicado o artigo como resultado deste trabalho: BUSTAMANTE, M. V. C.; FARIA, G. G.; ALVES, M. V. A.; BERTOTI, Giuliano Araujo . **APLICAÇÃO DE REALIDADE AUMENTADA PARA DISPOSITIVOS MÓVEIS.** Boletim Técnico da Faculdade de Tecnologia de São Paulo, v. 40, p. 101, 2015.

Secundariamente, foi publicado o seguinte artigo correlato: ALVES, M. V. A.; FARIA, G. G.; BUSTAMANTE, M. V. C.; BERTOTI, Giuliano Araujo . **MAGIC POKER: JOGO**

PARA DISPOSITIVOS MÓVEIS UTILIZANDO INTERFACES NATURAIS. Boletim Técnico da Faculdade de Tecnologia de São Paulo, v. 40, p. 112, 2015. A aplicação denominada MagicPoker está relacionada com este trabalho por utilizar uma plataforma totalmente web, que pode ser acessada em qualquer dispositivo móvel independente do sistema operacional, e também por utilizar Interfaces Naturais.

5.2 – Trabalhos Futuros

Após a conclusão deste, foram abertas as seguintes possibilidades de trabalhos futuros:

- Desenvolvimento de bibliotecas *web* que facilitem a criação de aplicações de realidade aumentada;
- Uso de Interfaces Naturais para melhorar a experiência em aplicações já existentes;
- Migração deste aplicativo para plataformas nativas (Android e IOS), para melhorar funcionalidades e desempenho.
- Aplicação de mapeamento de pontos turísticos, utilizando realidade aumentada para visualizar informações sobre os pontos;
- Jogo utilizando realidade aumentada para inserir personagens no mundo real;
- Aplicação de Realidade Virtual para plataforma Web.

REFERÊNCIAS

ANDREWS, Keith Michael; YIM, Philip. **Command line interface**. U.S. Patent Application 13/937,665, 9 jul. 2013.

Angularjs.org. **AngularJS - Superheroic JavaScript MVW Framework.** Disponívels em https://angularjs.org/. Acessado em: 03/03/2015

ARAÚJO, R. B. **Especificação e análise de um sistema distribuído de realidade virtual**, São Paulo, Junho, Tese (Doutorado), Departamento de Engenharia de Computação e Sistemas Digitais, Escola Politécnica da Universidade de São Paulo, 1996.

AUKSTAKALNIS, Steve; MIRAGE, David Blatner, Silicon. **The Art and Science of Virtual Reality**, Peachpit Press, Berkeley, CA, 1992.

AZUMA, Ronald T. **A survey of augmented reality**. Presence, v. 6, n. 4, p. 355-385, 1997.

AZUMA, Ronald. **Recent advances in augmented reality. Computer Graphics and** Applications, IEEE, v. 21, n. 6, p. 34-47, 2001.

CLARKE, Arthur C**. Hazards of prophecy: the failure of imagination. Profiles of the Future**, Gollancz, London, 1962.

FLANAGAN, David. **JavaScript: the definitive guide**. " O'Reilly Media, Inc.", 2006.

FRAIN, Ben. **Responsive web design with HTML5 and CSS3**. Packt Publishing Ltd, 2012.

FRITZ, Daniel; MOSSEL, Annette; KAUFMANN, Hannes. **Markerless 3D Interaction in an Unconstrained Handheld Mixed Reality Setup**; accepted for publication in The International Journal of Virtual Reality (2015).

HALES, Wesley. **HTML5 and JavaScript Web Apps**. " O'Reilly Media, Inc.", 2012.

HOLDENER, Anthony T. **HTML5 Geolocation**. " O'Reilly Media, Inc.", 2011.

KASAHARA, Shunichi; HEUN, Valentin; LEE, Austin S.; ISHII, Hiroshi. **Second Surface: Multi-user Spatial Collaboration System based on Augmented Reality**, SIGGRAPH Asia 2012 Emerging Technologies.

LIU, Weiyuan. **Natural user interface- next mainstream product user interface**, Computer-Aided Industrial Design & Conceptual Design (CAIDCD), 2010 IEEE 11th International Conference on , vol.1, no., pp.203-205, 17-19 Nov. 2010.

LOPPINI, Fabrizio; BIANCHINI, Paolo. **Graphical user interface**. U.S. Patent Application 09/846,572, 1 maio 2001.

Mark Otto, a. **Bootstrap - The world's most popular mobile-first and responsive front-end framework**. Disponível em http://getbootstrap.com/. Acesso em: 03/08/2015.

MICHELIS, Giorgio de; PAOLI, Flavio de; PLUCHINOTTA, Costanza; SUSANI, Marco. **Weakly augmented reality: observing and designing the work-place of creative designers**. Proceedings of DARE 2000 on Designing augmented reality environments, p.81-91, April 2000, Elsinore, Denmark.

MILGRAM, Paul et al. **Augmented reality: A class of displays on the reality-virtuality continuum.** In: Photonics for Industrial Applications. International Society for Optics and Photonics, 1995. p. 282-292.

MIT, 2015. Disponível em: http://indistinguishablefrom.media.mit.edu/syllabus/ (acesso em 19/05/2015).

PILGRIM, Mark. **HTML5: up and running**. O'Reilly Media, Inc.", 2010.

REKIMOTO, Jun; AYATSUKA, Yuji. **CyberCode: designing augmented reality environments with visual tags**, Proceedings of DARE 2000 on Designing augmented reality environments, p.1-10, April 2000, Elsinore, Denmark.

SUTHERLAND, I.E. (1965). **The Ultimate Display**. In Proceedings of IFIPS Congress, New York City, NY, vol. 2, p. 506-508.

SUTHERLAND, I.E. (1968). **A Head-mounted Three-dimensional Display**. In: Fall Joint Computer Conference, AFIPS Conference Proceedings, vol. 33, p. 757-764.

TORI, Romero; KIRNER, Claudio; SISCOUTTO, Robson Augusto. **Fundamentos e tecnologia de realidade virtual e aumentada**. Editora SBC, 2006.

VENDITTI, Benjamin. **Interaction in Dense One-Handed Handheld Augmented Reality,** Supervisor: H. Kaufmann, A. Mossel; Insitute for Software Technology and Interactive Systems, 2015; final examination: 01-15-2015.