Autonomous Tour Guide Final Package

Project group 6

Sindhu Reddy Golconda - 8
Simon Moeller - 15
Prudhvi Raj Mudunuri - 16
Sudhakar Reddy Peddinti - 21

University of Missouri - Kansas City
CS5542
May 10, 2016

Table of Contents

Project Summary

Introduction

The Autonomous Tour Guide is envisioned as a system that can provide a user with relative directions from the user's current location based on images taken by an autonomous robot. The robot would send a stream of images to the backend processor, and after processing the photos, the user would be able to ask for directions to a desired location, store, object, etc. If the desired location was identified in the images, the user would be provided with relative directions from the current location to the desired location. For example, if the desired location is on the user's right and behind the drinking fountain, the directions would tell the user to turn right and proceed past the drinking fountain.

The motivation for the Autonomous Tour Guide is to provide yet another way for persons with mobility concerns to maintain a more active lifestyle. In large, enclosed areas like shopping malls, GPS directions are generally not available. A user may be forced to wander for a distance before finding what he or she desires, which can be burdensome for a person with limited mobility. The Autonomous Tour Guide allows the user to travel directly to the desire without wasting energy searching.

Solutions

This project focuses entirely on image processing. As such, multiple algorithms are utilized to extract features from images, including SURF and ORB, and RANSAC for identify primary features. The intention would be for the real time data to be streamed from a robot, but for this project stock photos collected from Google Images were used in all cases as a substitute. All stock input images are automatically resized to a uniform image height before attempting any feature identifications. The stitching process is one of the most time intensive processes, and as such performance metrics were gathered for this process. The time to stitch images grew linearly with the size of the images, and full details can be found in the presentation at the end of this document.

Random Forest trees are used to for the machine learning. The trees were trained and tested using collections of images gathered from Google Images, and manually labeled for the training. Random forest trees worked well when identifying objects in images, achieving an 87.5% success rate with the sample data. Optical Character Recognition was less successful, achieving only a 9.1% complete success rate (key text fully recognized), and a 27.3% partial success rate (at least part of the key text recognized).

Future Work

This project included a very large feature set, and as such there is room for many future updates. Some of the features that could be included in the future:

- Provide a full smartphone/smartwatch based user GUI
- Support full image streaming at the backend server process, rather than static images
- Include a robot with full controls and navigation
- Build larger training/testing datasets to improve the real world accuracy of object detection and optical character recognition
- Relative image distance detection in 2D image (investigations had been done, but no solution identified)

Project artifacts

- This PDF - contains Project Proposal, Increment 1, Increment 2, Increment 3, and Increment 4 papers
- Youtube demo video: https://youtu.be/5vYRP4E04IU
- Github: https://github.com/SCE-UMKC/BigData-Spring2016-TourGuide
- Project final presentation: at the end of this document

Project Management

All members of the project team worked well together. There was good communication using WhatsApp and email, with collaboration managed through Google Drive and Github. In addition, the team would meet once per week on campus to work together and sync up. The work was divided based on feature.

Individual efforts and scores (note that not all features worked in earlier iterations were utilized in the final product):

- Golconda, Sindhu Reddy - 8
  - 25 points
  - Image resizing, relative image distance detection, object identification, speech to text
- Moeller, Simon - 15
  - 25 points
  - Image stitching, object identification, REST interface
- Mudunuri, Prudhvi Raj - 16
  - 25 points
  - Optical character recognition, Android image capture, object identification
- Peddinti, Sudhakar Reddy - 21
  - 25 points
  - Optical character recognition, Android image capture, object identification

Final Project Evaluation

The team held a project review meeting, and the general consensus was that the project was a much bigger undertaking than we anticipated. We feel we could have produced a more complete product had we focused on two or three key features, and built those features out to completion. Instead, we added four new features with every increment, but never felt that any achieved a final polished feel. In addition, the constant addition of new features caused the focus of the project to change over time, resulting in many of the features worked earlier being dropped from the final product.

Autonomous Tour Guide Proposal

Project group 6

Sindhu Reddy Golconda - 8
Simon Moeller - 15
Prudhvi Raj Mudunuri - 16
Sudhakar Reddy Peddinti - 21

University of Missouri - Kansas City
CS5542
January 27, 2016

The project proposal for group 6 is for a robot that can autonomously discover the layout of a new location, and then act as a guide for the human wearing the linked smart watch. This would be a take on an autonomous search and rescue or reconnaissance drone. The idea would be that the controller would wear the linked smartwatch, release the robot into a new building or location to search and map. After sufficient time the controller would be able to ask the watch to take him or her to a particular location within the building. The robot would then find its way back to the user, and would lead the user to the desired location.

The robot would be controlled by a smartphone. The smartphone would send a continuous stream of images to the backend analytics server. The backend analytics server would attempt to compare these images to online image repositories in an attempt to identify the objects and locations in the images. As objects and locations are identified, their location will be saved. After a sufficient area has been mapped, and some objects and locations have been identified, the user's smartwatch will be notified.

Once notified that the area has been sufficiently mapped, the user can verbally ask the smartwatch to take the user to one of the identified objects or locations. The smartwatch will use a direct link to talk with the robot using wifi or bluetooth or some other useable link. Using location and signal strength information, the robot will attempt to find the wearer of the smartwatch. Once the robot has located the wearer of the smartwatch, the robot will audibly inform the user. Once the user then gives the verbal command to start, the robot will begin to lead the user to the desired object or location within the building.

An example could be to utilize the robot to find a trapped person within an office building after an earthquake. The robot could be released to map the remaining structure. If a human body is identified, the user could ask the robot to lead the user and a rescue team to the human body. Or a more simplified example could be to release the robot into a new house, and then have the robot lead the user to the refrigerator so the user could get a drink.

This project is a very large and complex undertaking, and it is anticipated that it will not be able to be completed in full by the end of the semester. However, this project will allow the team to gain exposure to numerous new technologies with a focus on big data analytics and wearable device and robotics integrations. The work will be prioritized to focus on the important big data analytics and device integrations, with additional features given lower priority.

Bibliography:

Autonomous Tour Guide
Increment 1

Project group 6

Sindhu Reddy Golconda - 8
Simon Moeller - 15
Prudhvi Raj Mudunuri - 16
Sudhakar Reddy Peddinti - 21

University of Missouri - Kansas City

CS5542
February 19, 2016

Introduction

The project for group 6 is for a robot that can autonomously discover the layout of a new location, and then act as a guide for the human wearing the linked smart watch. This is a take on an autonomous search and rescue or reconnaissance drone. The idea is that the controller would wear the linked smartwatch, release the robot into a new building or location to search and map. After sufficient time the controller can ask the watch to take him or her to a particular location within the building. The robot will then find its way back to the user, and will lead the user to the desired location.

The robot is controlled by a smartphone. The smartphone sends a continuous stream of images to the backend analytics server. The backend analytics server compares these images to a database of known objects. As objects and locations are identified, their location is saved. After a sufficient area is mapped, with some objects and locations identified, the user's smartwatch is notified.

Once notified that the area is sufficiently mapped, the user will verbally ask the smartwatch to take the user to one of the identified objects or locations. The smartwatch uses a direct link to talk with the robot using wifi or bluetooth or some other useable link. Using location and signal strength information, the robot attempts to find the wearer of the smartwatch. Once the robot locates the wearer of the smartwatch, the robot audibly informs the user. The user then gives the verbal command to start, and the robot will lead the user to the desired object or location within the building.

An example would be to utilize the robot to assist a person with limited mobility to find the best route to a desired location within a new structure. The robot would be able to map the inside of the structure, and when instructed, lead the user to the desired location using a path that adheres to the user's particular restrictions.

Project Goal and Objectives

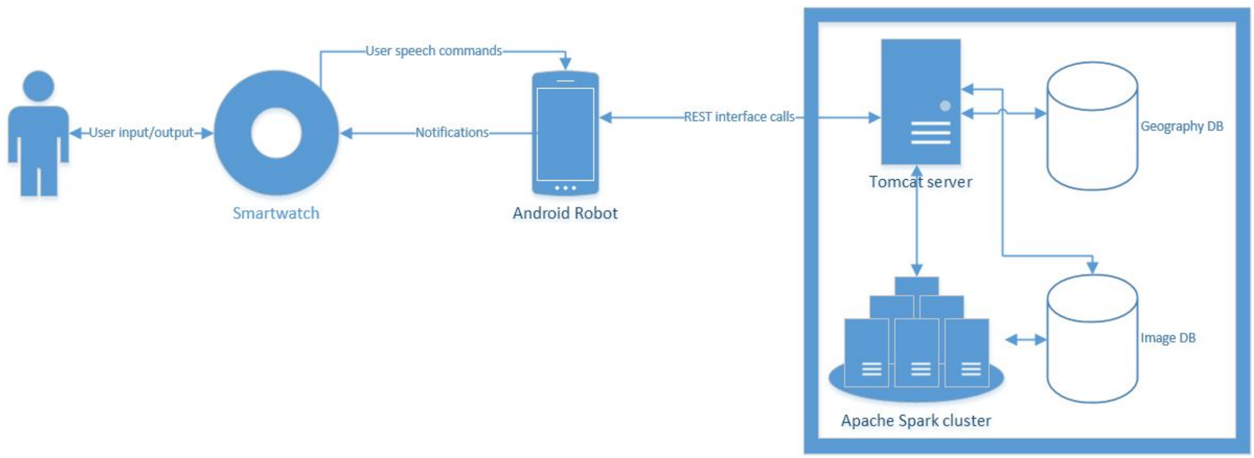
This project is a very large and complex undertaking, and as such the features of the project have been prioritized to favor primary learning goals for the team. Many teams have spent countless thousands of hours attempting to build robotic guides similar to what what is proposed here. Due to the level of difficulty in producing a fully functional autonomous tour guide robot, the desired features for this project have been stack ranked. The more critical learning objectives are ranked higher, and less critical learning objectives are ranked lower.

Project goals:

- Stack rank High
    - Speech to text through smartwatch, for sending commands to the smartphone controlled robot
    - REST communications interface between smartphone and backend analytics server
    - Capture and send images from camera to backend analytics server at beneficial intervals
    - Backend analytics server to process image identification against each image received
    - Backend tracking of location of identified images in relation to the robot as it moves
    - Provide updates to the smartwatch each time a new image is identified
    - Use a MapReduce based algorithm on Apache Spark for image detection
- Stack rank Medium
    - Identify and track the direction and distance from the camera for each identified image
    - Self learning what images should look like, based on a user provided list of objects to expect
    - Be able to retrace steps to find a previously identified image
- Stack rank Low
    - Support the full range of robot movement (ranked low due to the known defects with the API)
    - Obstacle avoidance when the robot moves
    - High success rate of accurate image detection
    - Have the robot find the wearer of the smartwatch based on available methods
    - Identify and use shortest path between points when the robot either finds the smartwatch or finds a previously identified image

Design

The design of the autonomous tour guide is based on four primary components. These are the user's smartwatch, the Android powered robot, the backend Tomcat server, and the backend Apache Spark cluster. The user's smartwatch is the primary user interface device, providing a speech to text capability and user notifications. The Android robot is responsible for physically navigating about the area, and sending a regular stream of images with geolocation information to the backend systems. The Tomcat server is responsible for keeping a record of the geolocation information received from the robot, and provides the "brains" for calculating movement commands to direct the robot. The Apache Spark cluster is responsible for image calculations, including keypoint identification and similarity scoring between images.



Individual components that have already been worked are described below.

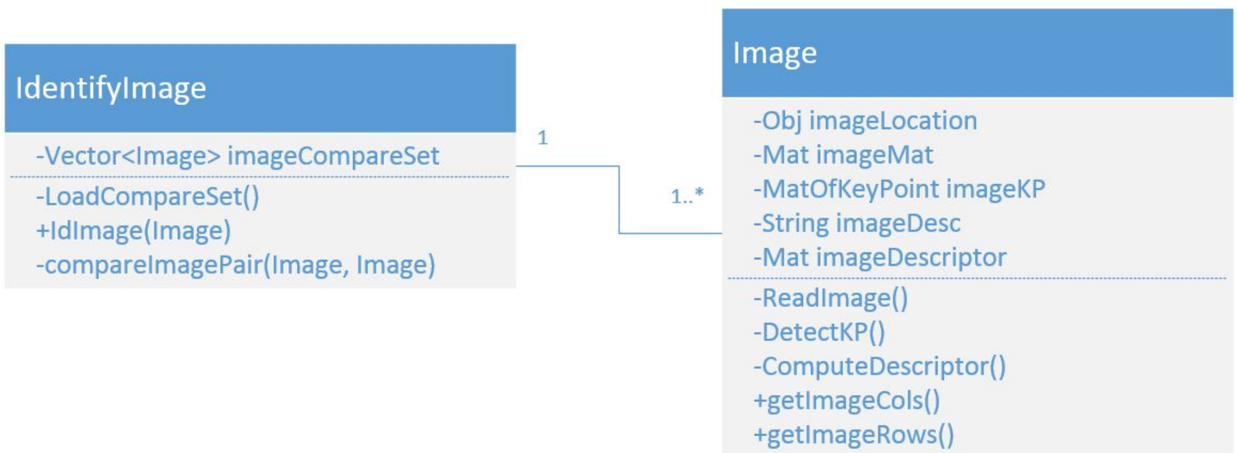
Image matching engine

Source code repository:
https://github.com/smoeller1/BigData-Spring2016-TourGuide/tree/master/src/ImageId/src/ImageId

The image matching engine is written as Java classes utilizing libraries from the OpenCV project. The first class is the Image class, responsible for identifying and saving the key points for the image. The second class is the IdentifyImage class, responsible for comparing different Image's to each other.

Each individual image is saved as an Image object. Upon instantiation, the Image object reads in the image file using the appropriate method depending on the image source location. Next, the OpenCV FeatureDetector is used to identify the key feature points using the ORB method, though other methods can be utilized. Last, the OpenCV DescriptorExtractor extracts the descriptors of the image using the ORB method.

The IdentifyImage class automatically creates Image objects for each of the comparison images upon instantiation. The goal is to have the comparison images be self-learned based on a keyword list of images that should be learned. However, at this time the list of comparison images is hard coded into the LoadCompareSet method. When a new image needs to be identified, the calling program calls the IdImage method with the new image that needs to be identified. The IdImage method will compare this new image against the database of known comparison images to attempt to find a match. The image comparison is done using the OpenCV Imgproc MatchTemplate to attempt to identify subsections of the new image, when possible.



Speech to text

Source code repository: https://github.com/SindhuReddyG-sgdd7/SpeechToText

The speech to text functionality utilizes the microphone on the smartwatch to allow the user to speak commands to the Android robot. The speech to text allows for text based commands to be easily processed by the Android robot.

REST Interface

Source code repository:
https://github.com/smoeller1/BigData-Spring2016-TourGuide/tree/master/src/ImageId/src/ImageId

The REST interface uses JSON formatted data to communicate between the Android robot and the backend servers. This interface is used for sending the images captured by the Android robot, as well as receiving messages from the servers when items are identified within the images, so that this information can be relayed on to the smartwatch.

Camera controller

Source code repository:
https://github.com/smoeller1/BigData-Spring2016-TourGuide/tree/master/src/ImageId/src/ImageId

The camera controller works on both a timer and a movement tracker. Each time the timer counts down, the camera controller will check to see if there has been movement since the last image was captured. If there has been movement, a new image is captured. If there has not been movement, then no image is captured and the timer is reset.

External APIs

OpenCV - Image processing libraries. http://opencv.org

Project Plan

The project is divided into four iterations. The iterations are described in more detail below. In general, with each sprint each team member is focusing on finishing a single feature. At the start of each sprint the remaining features are divided among the team members, with all team members working on all parts of the system over the course of the project.

Iteration 1

The four features that were worked during iteration 1 were speech to text, REST communications interface, camera controls, and image identification. These four features are described in more detail above. The work this sprint was completed by the following team members:

Sindhu Golconda - Speech to text
Simon Moeller - Image identification
Prudhvi Mudunuri - Camera controller
Sudhakar Peddinti - REST communications interface

We attempted to use ZenHub to help track work, however ZenHub went down (or just stopped working) for all team members partway through the iteration. Since the team members already understood what needed to be done, we continued to work without ZenHub.

All team members communicated well during the iteration, and the team met in person multiple times over the course of the iteration. This good communication also helped the team to largely achieve the goals of the iteration. There are a few minor defects still being resolved in some of the code from this iteration, but it is anticipated that these debugging efforts will be complete by this weekend.

Bibliography

OpenCV libraries. http://opencv.org

Autonomous Tour Guide
Increment 2

Project group 6

Sindhu Reddy Golconda - 8
Simon Moeller - 15
Prudhvi Raj Mudunuri - 16
Sudhakar Reddy Peddinti - 21

University of Missouri - Kansas City
CS5542
March 11, 2016

Original project introduction

The project for group 6 is for a robot that can autonomously discover the layout of a new location, and then act as a guide for the human wearing the linked smart watch. This is a take on an autonomous search and rescue or reconnaissance drone. The idea is that the controller would wear the linked smartwatch, release the robot into a new building or location to search and map. After sufficient time the controller can ask the watch to take him or her to a particular location within the building. The robot will then find its way back to the user, and will lead the user to the desired location.

The robot is controlled by a smartphone. The smartphone sends a continuous stream of images to the backend analytics server. The backend analytics server compares these images to a database of known objects. As objects and locations are identified, their location is saved. After a sufficient area is mapped, with some objects and locations identified, the user’s smartwatch is notified.

Once notified that the area is sufficiently mapped, the user will verbally ask the smartwatch to take the user to one of the identified objects or locations. The smartwatch uses a direct link to talk with the robot using wifi or bluetooth or some other useable link. Using location and signal strength information, the robot attempts to find the wearer of the smartwatch. Once the robot locates the wearer of the smartwatch, the robot audibly informs the user. The user then gives the verbal command to start, and the robot will lead the user to the desired object or location within the building.

An example would be to utilize the robot to assist a person with limited mobility to find the best route to a desired location within a new structure. The robot would be able to map the inside of the structure, and when instructed, lead the user to the desired location using a path that adheres to the user’s particular restrictions.

Updated project introduction

The scope of the Autonomous Tour Guide project has been revised after the feedback from increment 1. As such, this document has been updated to reflect the revised project.

The Autonomous Tour Guide is now envisioned as a system that can provide a user with relative directions from the user's current location based on images taken by the user's Android smartphone. The user would take a 360 degree series of photos using the Android smartphone, and after processing the photos, the user would be able to ask for directions to a desired location, store, object, etc. If the desired location was identified in the images, the Android smartphone will provide relative directions from the current location to the desired location. For example, if the desired location is on the user's right and behind the drinking fountain, the smartphone would tell the user to turn right and proceed past the drinking fountain.

The Android smartphone would take photos as the user aims it around the user's current location. The photos would be sent to a backend Apache Spark server, where they will be combined into a full panoramic. Once the panoramic is created, the Apache Spark server will then analyze the image to identify any objects that can be identified, as well as relative positions of the objects. Relative depth will be determined based on scaling of the identified objects in relation to the scaling of the other surrounding identified objects.

Once all objects are identified and relative positions ascertained, the Android smartphone will be notified. The user will then be able to ask for directions to any of the identified objects. The request for directions will again be sent to the backend server for processing, and the computed relative directions will then be sent back to the Android smartphone for end user consumption.

Due to the nature of this project, all libraries are either C++ wrapped in Java, or native Java. The Spark implementation is written in Scala, though it will be very similar to a Java implementation due to the Java libraries used.

Project Goal and Objectives

This project is a very large and complex undertaking, and as such the features of the project have been prioritized to favor primary learning goals for the team. Many teams have spent countless thousands of hours attempting to build robotic guides similar to what what is proposed here. Due to the level of difficulty in producing a fully functional autonomous tour guide robot, the desired features for this project have been stack ranked. The more critical learning objectives are ranked higher, and less critical learning objectives are ranked lower.

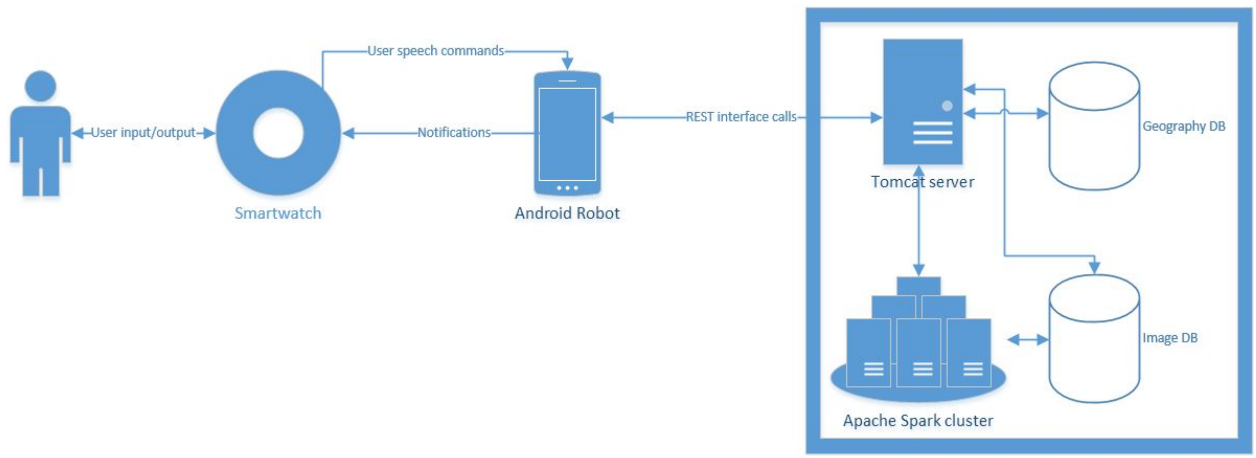
Project goals:

- Stack rank High
    - Object identification within images (no scaling)
    - Image cascade identification (includes scaling)
    - Object relative depth identification
    - Image stitching
    - Speech to text through smartwatch, for sending commands to the smartphone controlled robot
    - REST communications interface between smartphone and backend analytics server
    - Capture and send images from camera to backend analytics server at beneficial intervals
    - Backend analytics server to process image identification against each image received
    - Backend tracking of location of identified images in relation to the robot as it moves
    - Optical character recognition within a photograph
    - Use a MapReduce based algorithm on Apache Spark for image detection
- Stack rank Medium
    - Identify and track the direction and distance from the camera for each identified image
    - Self learning what images should look like, based on a user provided list of objects to expect
    - Be able to retrace steps to find a previously identified image
- Stack rank Low
    - Support the full range of robot movement (ranked low due to the known defects with the API)
    - Provide updates to the smartwatch each time a new image is identified
    - Obstacle avoidance when the robot moves
    - High success rate of accurate image detection
    - Have the robot find the wearer of the smartwatch based on available methods

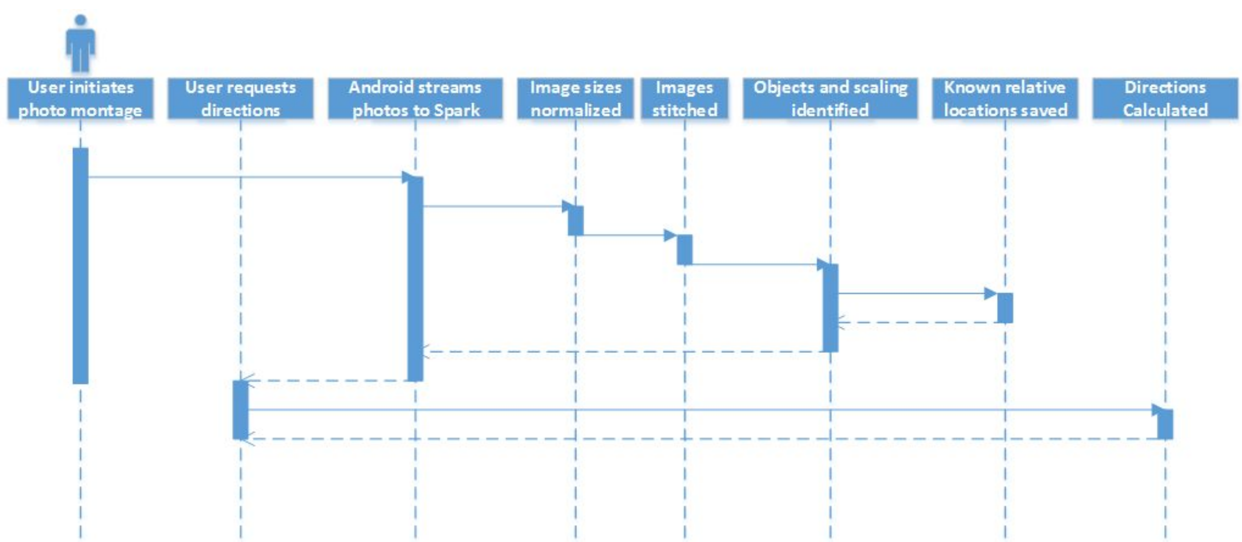
- Identify and use shortest path between points when the robot either finds the smartwatch or finds a previously identified image

Design

The design of the autonomous tour guide is based on four primary components. These are the user's smartwatch, the Android powered robot, the backend Tomcat server, and the backend Apache Spark cluster. The user's smartwatch is the primary user interface device, providing a speech to text capability and user notifications. The Android robot is responsible for physically navigating about the area, and sending a regular stream of images with geolocation information to the backend systems. The Tomcat server is responsible for keeping a record of the geolocation information received from the robot, and provides the "brains" for calculating movement commands to direct the robot. The Apache Spark cluster is responsible for image calculations, including keypoint identification and similarity scoring between images.



Basic design



High level sequence diagram

Individual components that have already been worked are described below.

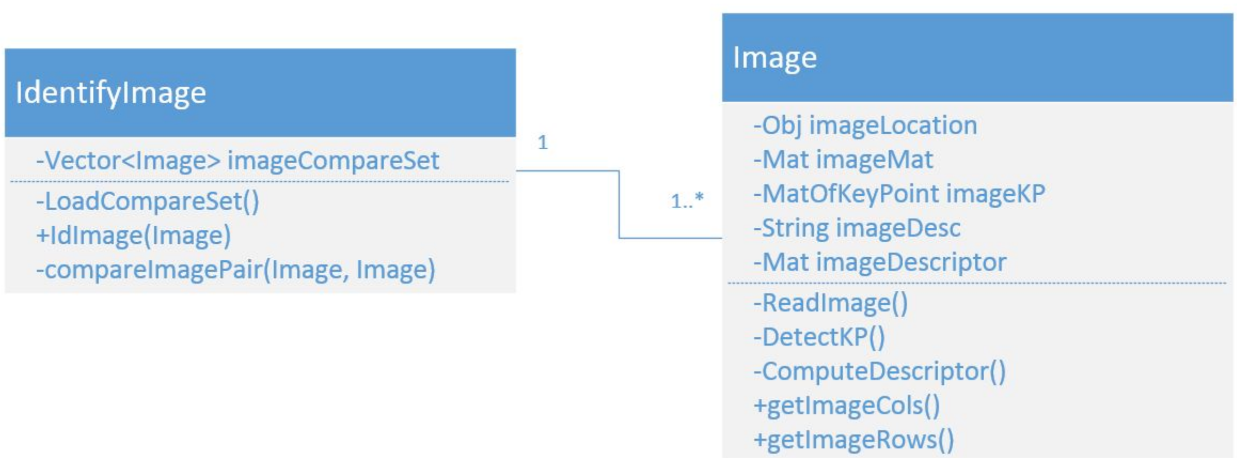
Image matching engine

Source code repository:
https://github.com/smoeller1/BigData-Spring2016-TourGuide/tree/master/src/ImageId/src/ImageId

The image matching engine is written as Java classes utilizing libraries from the OpenCV project. The first class is the Image class, responsible for identifying and saving the key points for the image. The second class is the IdentifyImage class, responsible for comparing different Image's to each other.

Each individual image is saved as an Image object. Upon instantiation, the Image object reads in the image file using the appropriate method depending on the image source location. Next, the OpenCV FeatureDetector is used to identify the key feature points using the ORB method, though other methods can be utilized. Last, the OpenCV DescriptorExtractor extracts the descriptors of the image using the ORB method.

The IdentifyImage class automatically creates Image objects for each of the comparison images upon instantiation. The goal is to have the comparison images be self-learned based on a keyword list of images that should be learned. However, at this time the list of comparison images is hard coded into the LoadCompareSet method. When a new image needs to be identified, the calling program calls the IdImage method with the new image that needs to be identified. The IdImage method will compare this new image against the database of known comparison images to attempt to find a match. The image comparison is done using the OpenCV Imgproc MatchTemplate to attempt to identify subsections of the new image, when possible.



Speech to text

Source code repository: https://github.com/SindhuReddyG-sgdd7/SpeechToText

The speech to text functionality utilizes the microphone on the smartwatch to allow the user to speak commands to the Android robot. The speech to text allows for text based commands to be easily processed by the Android robot.

REST Interface

Source code repository:
https://github.com/smoeller1/BigData-Spring2016-TourGuide/tree/master/src/ImageId/src/ImageId

The REST interface uses JSON formatted data to communicate between the Android robot and the backend servers. This interface is used for sending the images captured by the Android robot, as well as receiving messages from the servers when items are identified within the images, so that this information can be relayed on to the smartwatch.

Camera controller

Source code repository:
https://github.com/smoeller1/BigData-Spring2016-TourGuide/tree/master/src/ImageId/src/ImageId

The camera controller works on both a timer and a movement tracker. Each time the timer counts down, the camera controller will check to see if there has been movement since the last image was captured. If there has been movement, a new image is captured. If there has not been movement, then no image is captured and the timer is reset.

Image normalization

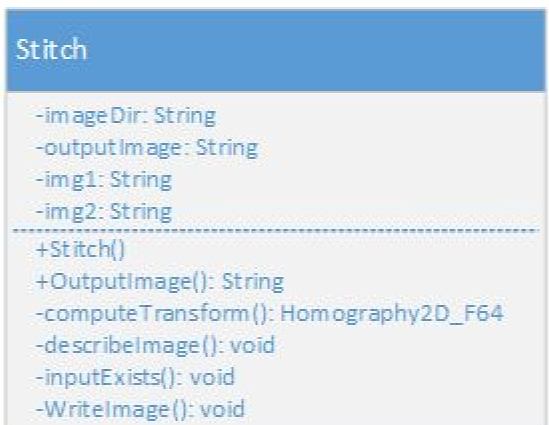
Source code repository: https://github.com/SindhuReddyG-sgdd7/ImageScaling

As the images captured will be of different sizes, it is required to modify the pixels of all captured images to the same pixel intensity and same size. An Abstract Window Toolkit (AWT), a JAVA built-in tool is used which renders the outline of any image. All captured images will be given as input and the normalized images are stored in a new folder which are then processed for image stitching.

Image stitching

There are three basic ways to achieve image stitching in Java. The first way is to write entire implementation from scratch, implementing feature identification, feature matching, rotation, and scaling. This method, while a great learning experience, would take more time than is available in a single semester course. The second method would be to use the OpenCV C++ libraries, and compile them through a Java interface using SWIG or JavaCPP. This method is feasible, but does introduce added complexity to the code maintenance. The third method, which was chosen for this project, is to use the BoofCV libraries. These libraries provide some of the same features as OpenCV, but are written as native Java code.

When stitching a large number of images into a panoramic, the Stitching class can be parallelized in a binary tree fashion. On the first pass pairs of consecutive images will be stitched, with all pairs stitched concurrently (as far as hardware will allow). After the first pass there will be half as many images, which will be paired up again and stitched concurrently. This will continue until there is a single image left. The number of iterative parallel calls to Stitch would then be log(N).



Class diagram for Stitch class

The underlying algorithm that is used to identify the descriptors is the Speeded Up Robust Feature (SURF) algorithm. This algorithm is similar to the SIFT algorithm, but SURF is considered to be faster. The SURF algorithm has three main steps, interest point detection, local neighborhood description, and matching.

There is an outstanding defect with the image stitching class which is causing the stitched image to be created within a large, black image of static size. This will need to be addressed to have the stitched image size be based on the actual dimensions of the stitching, and have any additional space coded as white instead of black. This will also be useful when stitching images that have previously been stitched.

External APIs

OpenCV - C++ and Java Image processing libraries. http://opencv.org

BoofCV - Java Image processing libraries. http://boofcv.org

Project Plan

The project is divided into four iterations. The iterations are described in more detail below. In general, with each sprint each team member is focusing on finishing a single feature. At the start of each sprint the remaining features are divided among the team members, with all team members working on all parts of the system over the course of the project.

Iteration 1

The four features that were worked during iteration 1 were speech to text, REST communications interface, camera controls, and image identification. These four features are described in more detail above. The work this sprint was completed by the following team members:

Sindhu Golconda - Speech to text
Simon Moeller - Image identification
Prudhvi Mudunuri - Camera controller
Sudhakar Peddinti - REST communications interface

We attempted to use ZenHub to help track work, however ZenHub went down (or just stopped working) for all team members partway through the iteration. Since the team members already understood what needed to be done, we continued to work without ZenHub.

All team members communicated well during the iteration, and the team met in person multiple times over the course of the iteration. This good communication also helped the team to largely achieve the goals of the iteration. There are a few minor defects still being resolved in some of the code from this iteration, but it is anticipated that these debugging efforts will be complete by this weekend.

Iteration 2

The focus of iteration 2, based on feedback after iteration 1, was altered to focus more on image detection and manipulation processes. As such, new features were added such as optical character recognition, image normalization, and panoramic image stitching. Also based on the feedback from iteration 1, some of the original features have been lowered in priority. These include robot movement, some of the smartwatch interactions, and other features unrelated to direct image processing.

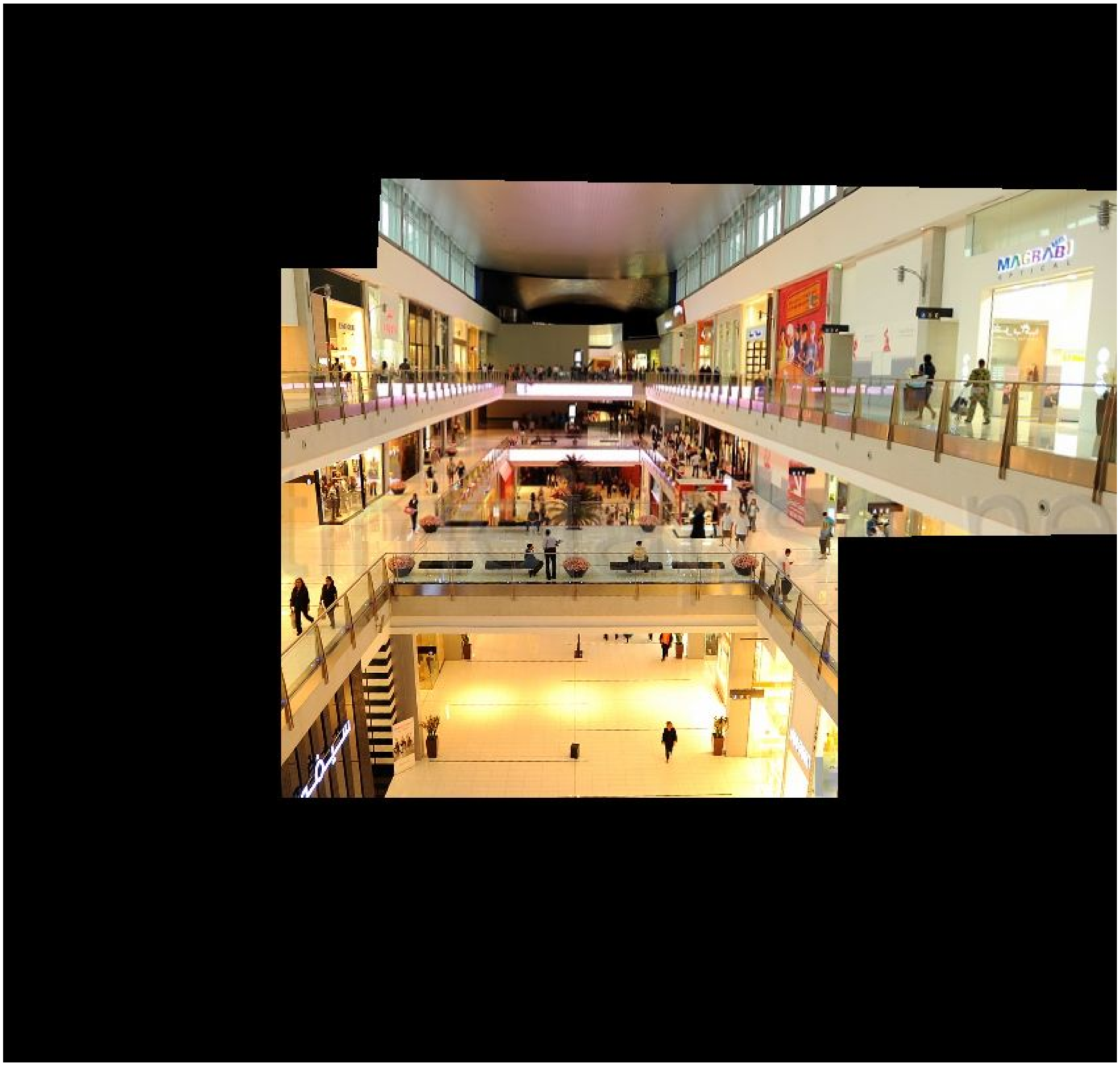
Sindhu Golconda - Image normalization
Simon Moeller - Image stitching
Prudhvi Mudunuri - Optical character recognition
Sudhakar Peddinti - Optical character recognition

The team met two to three times each week during iteration 2. The meetings were generally treating as status and design discussion sessions. After the change of direction for the project following the iteration 1 feedback, it did take a few team meetings before the new scope of the project was finalized. This meant that some additional time this iteration was spent on basic research tasks to identify what the new project scope could realistically be, and what the new priority of the features should be.

Example of image stitching

Input Image 1

Input image 2 (image scale is 0.5 of Input image 1)



Output stitched image

Bibliography

OpenCV libraries. http://opencv.org

BoofCV libraries. http://boofcv.org

Autonomous Tour Guide

Increment 3

Project group 6

Sindhu Reddy Golconda - 8
Simon Moeller - 15
Prudhvi Raj Mudunuri - 16
Sudhakar Reddy Peddinti - 21

University of Missouri - Kansas City

CS5542

April 6, 2016

Original project introduction

The project for group 6 is for a robot that can autonomously discover the layout of a new location, and then act as a guide for the human wearing the linked smart watch. This is a take on an autonomous search and rescue or reconnaissance drone. The idea is that the controller would wear the linked smartwatch, release the robot into a new building or location to search and map. After sufficient time the controller can ask the watch to take him or her to a particular location within the building. The robot will then find its way back to the user, and will lead the user to the desired location.

The robot is controlled by a smartphone. The smartphone sends a continuous stream of images to the backend analytics server. The backend analytics server compares these images to a database of known objects. As objects and locations are identified, their location is saved. After a sufficient area is mapped, with some objects and locations identified, the user's smartwatch is notified.

Once notified that the area is sufficiently mapped, the user will verbally ask the smartwatch to take the user to one of the identified objects or locations. The smartwatch uses a direct link to talk with the robot using wifi or bluetooth or some other useable link. Using location and signal strength information, the robot attempts to find the wearer of the smartwatch. Once the robot locates the wearer of the smartwatch, the robot audibly informs the user. The user then gives the verbal command to start, and the robot will lead the user to the desired object or location within the building.

An example would be to utilize the robot to assist a person with limited mobility to find the best route to a desired location within a new structure. The robot would be able to map the inside of the structure, and when instructed, lead the user to the desired location using a path that adheres to the user's particular restrictions.

Updated project introduction

The scope of the Autonomous Tour Guide project has been revised after the feedback from increment 1. As such, this document has been updated to reflect the revised project.

The Autonomous Tour Guide is now envisioned as a system that can provide a user with relative directions from the user's current location based on images taken by the user's Android smartphone. The user would take a 360 degree series of photos using the Android smartphone, and after processing the photos, the user would be able to ask for directions to a desired location, store, object, etc. If the desired location was identified in the images, the Android smartphone will provide relative directions from the current location to the desired location. For example, if the desired location is on the user's right and behind the drinking fountain, the smartphone would tell the user to turn right and proceed past the drinking fountain.

The Android smartphone would take photos as the user aims it around the user's current location. The photos would be sent to a backend Apache Spark server, where they will be combined into a full panoramic. Once the panoramic is created, the Apache Spark server will then analyze the image to identify any objects that can be identified, as well as relative positions of the objects. Relative depth will be determined based on scaling of the identified objects in relation to the scaling of the other surrounding identified objects.

Once all objects are identified and relative positions ascertained, the Android smartphone will be notified. The user will then be able to ask for directions to any of the identified objects. The request for directions will again be sent to the backend server for processing, and the computed relative directions will then be sent back to the Android smartphone for end user consumption.

Due to the nature of this project, all libraries are either C++ wrapped in Java, or native Java. The Spark implementation is written in Scala, though it will be very similar to a Java implementation due to the Java libraries used.

Project Goal and Objectives

This project is a very large and complex undertaking, and as such the features of the project have been prioritized to favor primary learning goals for the team. Many teams have spent countless thousands of hours attempting to build robotic guides similar to what what is proposed here. Due to the level of difficulty in producing a fully functional autonomous tour guide robot, the desired features for this project have been stack ranked. The more critical learning objectives are ranked higher, and less critical learning objectives are ranked lower.

Project goals:

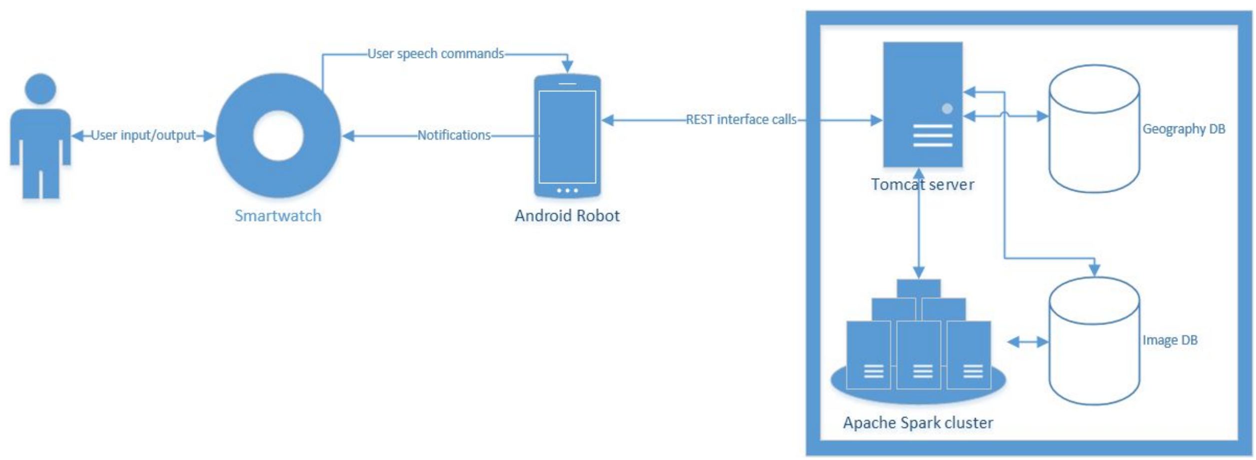
- Stack rank High
  - Object identification within images (no scaling)
  - Image cascade identification (includes scaling)
  - Object relative depth identification
  - Image stitching
  - Speech to text through smartwatch, for sending commands to the smartphone controlled robot
  - REST communications interface between smartphone and backend analytics server
  - Capture and send images from camera to backend analytics server at beneficial intervals
  - Backend analytics server to process image identification against each image received
  - Backend tracking of location of identified images in relation to the robot as it moves
  - Optical character recognition within a photograph
  - Use a MapReduce based algorithm on Apache Spark for image detection
- Stack rank Medium
  - Identify and track the direction and distance from the camera for each identified image
  - Self learning what images should look like, based on a user provided list of objects to expect
  - Be able to retrace steps to find a previously identified image
- Stack rank Low
  - Support the full range of robot movement (ranked low due to the known defects with the API)
  - Provide updates to the smartwatch each time a new image is identified
  - Obstacle avoidance when the robot moves
  - High success rate of accurate image detection
  - Have the robot find the wearer of the smartwatch based on available methods

- Identify and use shortest path between points when the robot either finds the smartwatch or finds a previously identified image

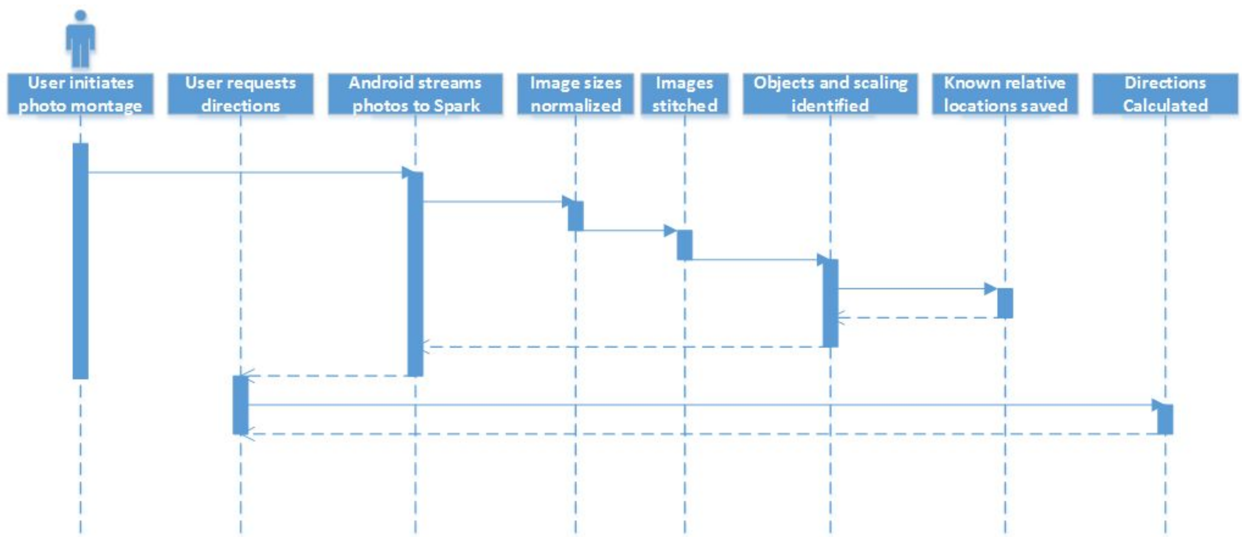
Design

TODO: Update anything that needs to be updated or added to the Design section for increment 3

The design of the autonomous tour guide is based on four primary components. These are the user's smartwatch, the Android powered robot, the backend Tomcat server, and the backend Apache Spark cluster. The user's smartwatch is the primary user interface device, providing a speech to text capability and user notifications. The Android robot is responsible for physically navigating about the area, and sending a regular stream of images with geolocation information to the backend systems. The Tomcat server is responsible for keeping a record of the geolocation information received from the robot, and provides the "brains" for calculating movement commands to direct the robot. The Apache Spark cluster is responsible for image calculations, including keypoint identification and similarity scoring between images.



Basic design



High level sequence diagram

Individual components that have already been worked are described below.

Image matching engine

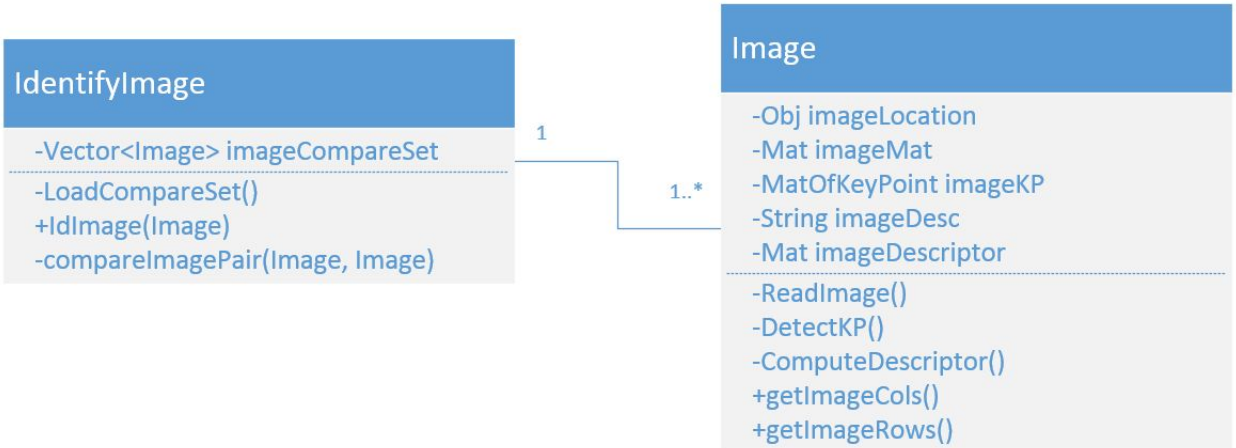
Source code repository:
https://github.com/smoeller1/BigData-Spring2016-TourGuide/tree/master/src/ImageId/src/ImageId

The image matching engine is written as Java classes utilizing libraries from the OpenCV project. The first class is the Image class, responsible for identifying and saving the key points for the image. The second class is the IdentifyImage class, responsible for comparing different Image's to each other.

Each individual image is saved as an Image object. Upon instantiation, the Image object reads in the image file using the appropriate method depending on the image source location. Next, the OpenCV FeatureDetector is used to identify the key feature points using the ORB method, though other methods can be utilized. Last, the OpenCV DescriptorExtractor extracts the descriptors of the image using the ORB method.

The IdentifyImage class automatically creates Image objects for each of the comparison images upon instantiation. The goal is to have the comparison images be self-learned based on a keyword list of images that should be learned. However, at this time the list of comparison images is hard coded into the LoadCompareSet method. When a new image needs to be identified, the calling program calls the IdImage method with the new image that needs to be identified. The IdImage method will compare this new image against the database of known comparison images to attempt to find a match. The image comparison is done using the

OpenCV Imgproc MatchTemplate to attempt to identify subsections of the new image, when possible.



Speech to text

Source code repository: https://github.com/SindhuReddyG-sgdd7/SpeechToText

The speech to text functionality utilizes the microphone on the smartwatch to allow the user to speak commands to the Android robot. The speech to text allows for text based commands to be easily processed by the Android robot.

REST Interface

Source code repository:
https://github.com/smoeller1/BigData-Spring2016-TourGuide/tree/master/src/ImageId/src/ImageId

The REST interface uses JSON formatted data to communicate between the Android robot and the backend servers. This interface is used for sending the images captured by the Android robot, as well as receiving messages from the servers when items are identified within the images, so that this information can be relayed on to the smartwatch.

Camera controller

Source code repository:
https://github.com/smoeller1/BigData-Spring2016-TourGuide/tree/master/src/ImageId/src/ImageId

The camera controller works on both a timer and a movement tracker. Each time the timer counts down, the camera controller will check to see if there has been movement since the last image was captured. If there has been movement, a new image is captured. If there has not been movement, then no image is captured and the timer is reset.

Image normalization

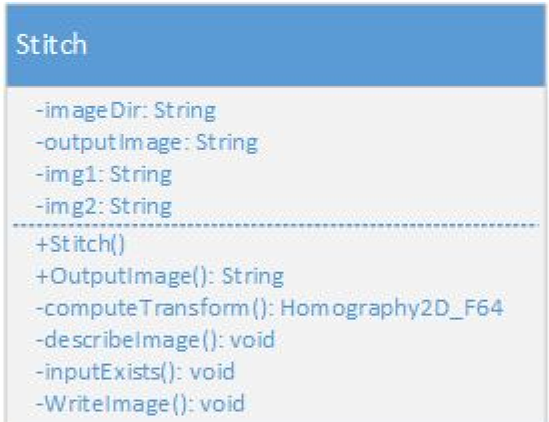
Source code repository: https://github.com/SindhuReddyG-sgdd7/ImageScaling

As the images captured will be of different sizes, it is required to modify the pixels of all captured images to the same pixel intensity and same size. An Abstract Window Toolkit (AWT), a JAVA built-in tool is used which renders the outline of any image. All captured images will be given as input and the normalized images are stored in a new folder which are then processed for image stitching.

Image stitching

There are three basic ways to achieve image stitching in Java. The first way is to write entire implementation from scratch, implementing feature identification, feature matching, rotation, and scaling. This method, while a great learning experience, would take more time than is available in a single semester course. The second method would be to use the OpenCV C++ libraries, and compile them through a Java interface using SWIG or JavaCPP. This method is feasible, but does introduce added complexity to the code maintenance. The third method, which was chosen for this project, is to use the BoofCV libraries. These libraries provide some of the same features as OpenCV, but are written as native Java code.

When stitching a large number of images into a panoramic, the Stitching class can be parallelized in a binary tree fashion. On the first pass pairs of consecutive images will be stitched, with all pairs stitched concurrently (as far as hardware will allow). After the first pass there will be half as many images, which will be paired up again and stitched concurrently. This will continue until there is a single image left. The number of iterative parallel calls to Stitch would then be log(N).



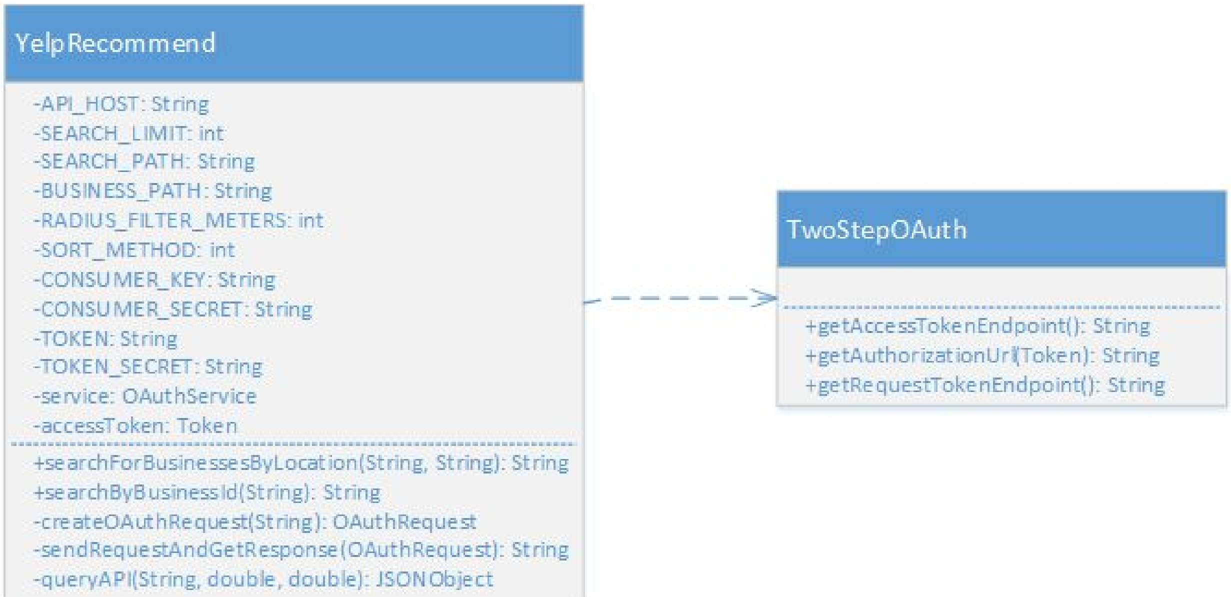
Class diagram for Stitch class

The underlying algorithm that is used to identify the descriptors is the Speeded Up Robust Feature (SURF) algorithm. This algorithm is similar to the SIFT algorithm, but SURF is considered to be faster. The SURF algorithm has three main steps, interest point detection, local neighborhood description, and matching.

There is an outstanding defect with the image stitching class which is causing the stitched image to be created within a large, black image of static size. This will need to be addressed to have the stitched image size be based on the actual dimensions of the stitching, and have any additional space coded as white instead of black. This will also be useful when stitching images that have previously been stitched.

Item Searching

Once an area is mapped, a user will want to be able to search for the best nearby business that can meet the shopping needs of the user. This nearby business searching is being provided by Yelp through the Yelp Developer API Version 2. This is a REST interface into the yelp system. By default the single highest rated nearby business that provides the item or service the user requested will be returned.



YelpRecommend class diagram

The Yelp API is accessed by OAuth v1.0a, with a set of authorization keys generated just for the Tour Guide application. By default the YelpRecommend class will restrict the search area to within 500 meters of the user, and will return the single highest rated business that meets the user's search criteria. According to Yelp, "highest rated" is based on an algorithm that incorporates both average user rating and number of ratings, in order to prevent a business with a single 5 star rating from dominating the returned results.

Sample Java usage:

```
JSONObject results = queryAPI((String) "Jeans", 38.952508, -94.719309);
System.out.println("Best match business: " + results.toString());
```

Results:

Best match business:
{"coordinate":{"latitude":"38.95641","longitude":"-94.71533"},"name":"Nordstrom"}

Github location: https://github.com/smoeller1/BigData-Spring2016-TourGuide

External APIs

OpenCV - C++ and Java Image processing libraries. http://opencv.org

BoofCV - Java Image processing libraries. http://boofcv.org

Yelp API - Reviews and rankings of nearby businesses.
https://www.yelp.com/developers/documentation/v2/overview

Project Plan

The project is divided into four iterations. The iterations are described in more detail below. In general, with each sprint each team member is focusing on finishing a single feature. At the start of each sprint the remaining features are divided among the team members, with all team members working on all parts of the system over the course of the project.

Iteration 1

The four features that were worked during iteration 1 were speech to text, REST communications interface, camera controls, and image identification. These four features are described in more detail above. The work this sprint was completed by the following team members:

Sindhu Golconda - Speech to text
Simon Moeller - Image identification
Prudhvi Mudunuri - Camera controller
Sudhakar Peddinti - REST communications interface

We attempted to use ZenHub to help track work, however ZenHub went down (or just stopped working) for all team members partway through the iteration. Since the team members already understood what needed to be done, we continued to work without ZenHub.

All team members communicated well during the iteration, and the team met in person multiple times over the course of the iteration. This good communication also helped the team to largely achieve the goals of the iteration. There are a few minor defects still being resolved in some of the code from this iteration, but it is anticipated that these debugging efforts will be complete by this weekend.

Iteration 2

The focus of iteration 2, based on feedback after iteration 1, was altered to focus more on image detection and manipulation processes. As such, new features were added such as optical character recognition, image normalization, and panoramic image stitching. Also based on the feedback from iteration 1, some of the original features have been lowered in priority. These include robot movement, some of the smartwatch interactions, and other features unrelated to direct image processing.

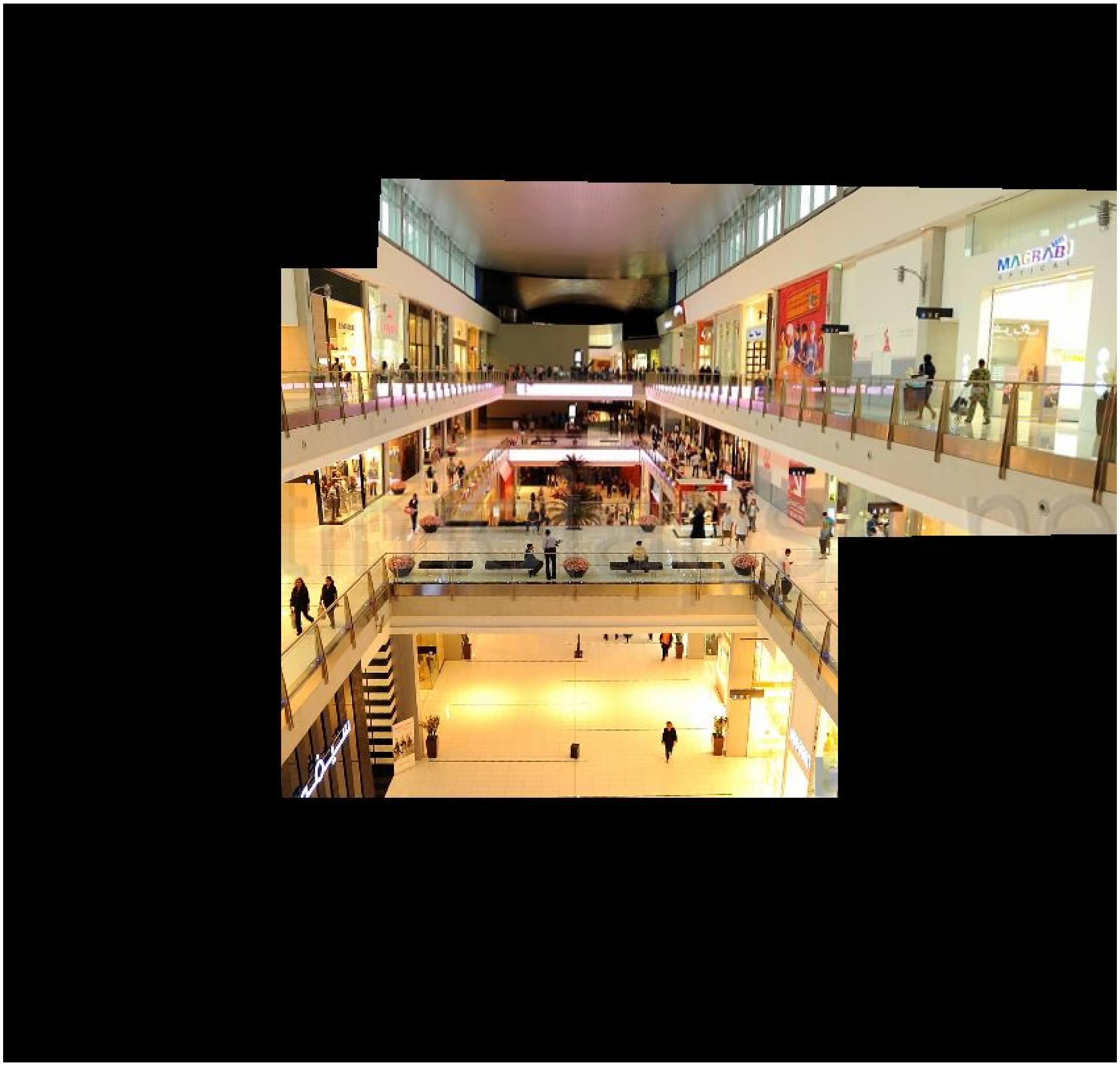
Sindhu Golconda - Image normalization
Simon Moeller - Image stitching
Prudhvi Mudunuri - Optical character recognition
Sudhakar Peddinti - Optical character recognition

The team met two to three times each week during iteration 2. The meetings were generally treating as status and design discussion sessions. After the change of direction for the project following the iteration 1 feedback, it did take a few team meetings before the new scope of the project was finalized. This meant that some additional time this iteration was spent on basic research tasks to identify what the new project scope could realistically be, and what the new priority of the features should be.

Example of image stitching

Input Image 1

Input image 2 (image scale is 0.5 of Input image 1)



Output stitched image

Iteration 3

TODO - Provide a brief overview of iteration 3

Sindhu Golconda - Distance estimation from an object
Simon Moeller - Business recommendations
Prudhvi Mudunuri - Object classification
Sudhakar Peddinti - Object classification

Sample recommendation: The user wishes to find a store that sells jeans near the current location. The current location is provided as GPS coordinates from the user's smartphone.

```
JSONObject results = queryAPI((String) "Jeans", 38.952508, -94.719309);
```

Results:

Nordstrom, located at 38.95641, -94.71533

Bibliography

OpenCV libraries. http://opencv.org

BoofCV libraries. http://boofcv.org

Yelp API V2. https://www.yelp.com/developers/

Autonomous Tour Guide
Increment 4

Project group 6

Sindhu Reddy Golconda - 8
Simon Moeller - 15
Prudhvi Raj Mudunuri - 16
Sudhakar Reddy Peddinti - 21

University of Missouri - Kansas City
CS5542
April 29, 2016

Original project introduction

The project for group 6 is for a robot that can autonomously discover the layout of a new location, and then act as a guide for the human wearing the linked smart watch. This is a take on an autonomous search and rescue or reconnaissance drone. The idea is that the controller would wear the linked smartwatch, release the robot into a new building or location to search and map. After sufficient time the controller can ask the watch to take him or her to a particular location within the building. The robot will then find its way back to the user, and will lead the user to the desired location.

The robot is controlled by a smartphone. The smartphone sends a continuous stream of images to the backend analytics server. The backend analytics server compares these images to a database of known objects. As objects and locations are identified, their location is saved. After a sufficient area is mapped, with some objects and locations identified, the user's smartwatch is notified.

Once notified that the area is sufficiently mapped, the user will verbally ask the smartwatch to take the user to one of the identified objects or locations. The smartwatch uses a direct link to talk with the robot using wifi or bluetooth or some other useable link. Using location and signal strength information, the robot attempts to find the wearer of the smartwatch. Once the robot locates the wearer of the smartwatch, the robot audibly informs the user. The user then gives the verbal command to start, and the robot will lead the user to the desired object or location within the building.

An example would be to utilize the robot to assist a person with limited mobility to find the best route to a desired location within a new structure. The robot would be able to map the inside of the structure, and when instructed, lead the user to the desired location using a path that adheres to the user's particular restrictions.

Updated project introduction

The scope of the Autonomous Tour Guide project has been revised after the feedback from increment 1. As such, this document has been updated to reflect the revised project.

The Autonomous Tour Guide is now envisioned as a system that can provide a user with relative directions from the user's current location based on images taken by the user's Android smartphone. The user would take a 360 degree series of photos using the Android smartphone, and after processing the photos, the user would be able to ask for directions to a desired location, store, object, etc. If the desired location was identified in the images, the Android smartphone will provide relative directions from the current location to the desired location. For example, if the desired location is on the user's right and behind the drinking fountain, the smartphone would tell the user to turn right and proceed past the drinking fountain.

The Android smartphone would take photos as the user aims it around the user's current location. The photos would be sent to a backend Apache Spark server, where they will be combined into a full panoramic. Once the panoramic is created, the Apache Spark server will then analyze the image to identify any objects that can be identified, as well as relative positions of the objects. Relative depth will be determined based on scaling of the identified objects in relation to the scaling of the other surrounding identified objects.

Once all objects are identified and relative positions ascertained, the Android smartphone will be notified. The user will then be able to ask for directions to any of the identified objects. The request for directions will again be sent to the backend server for processing, and the computed relative directions will then be sent back to the Android smartphone for end user consumption.

Due to the nature of this project, all libraries are either C++ wrapped in Java, or native Java. The Spark implementation is written in Scala, though it will be very similar to a Java implementation due to the Java libraries used.

Project Goal and Objectives

This project is a very large and complex undertaking, and as such the features of the project have been prioritized to favor primary learning goals for the team. Many teams have spent countless thousands of hours attempting to build robotic guides similar to what what is proposed here. Due to the level of difficulty in producing a fully functional autonomous tour guide robot, the desired features for this project have been stack ranked. The more critical learning objectives are ranked higher, and less critical learning objectives are ranked lower.

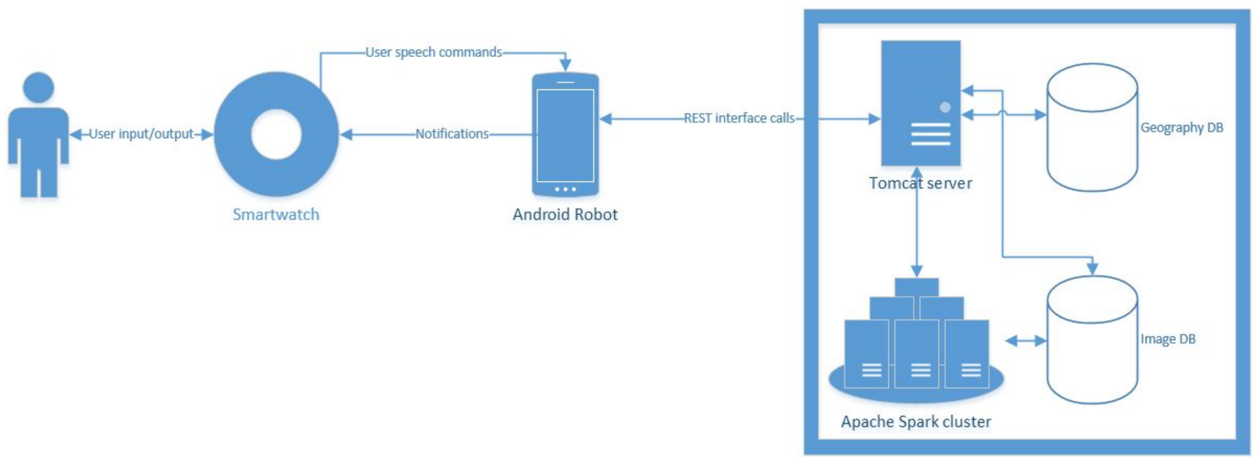
Project goals:

- Stack rank High
  - Object identification within images (no scaling)
  - Image cascade identification (includes scaling)
  - Object relative depth identification
  - Image stitching
  - Speech to text through smartwatch, for sending commands to the smartphone controlled robot
  - REST communications interface between smartphone and backend analytics server
  - Capture and send images from camera to backend analytics server at beneficial intervals
  - Backend analytics server to process image identification against each image received
  - Backend tracking of location of identified images in relation to the robot as it moves
  - Optical character recognition within a photograph
  - Use a MapReduce based algorithm on Apache Spark for image detection
- Stack rank Medium
  - Identify and track the direction and distance from the camera for each identified image
  - Self learning what images should look like, based on a user provided list of objects to expect
  - Gather information source (Yelp, Twitter, Instagram..) for recommendation system
  - Recommend the user with predicted business store based on the user provided input
  - Be able to retrace steps to find a previously identified image
- Stack rank Low
  - Support the full range of robot movement (ranked low due to the known defects with the API)
  - Provide updates to the smartwatch each time a new image is identified

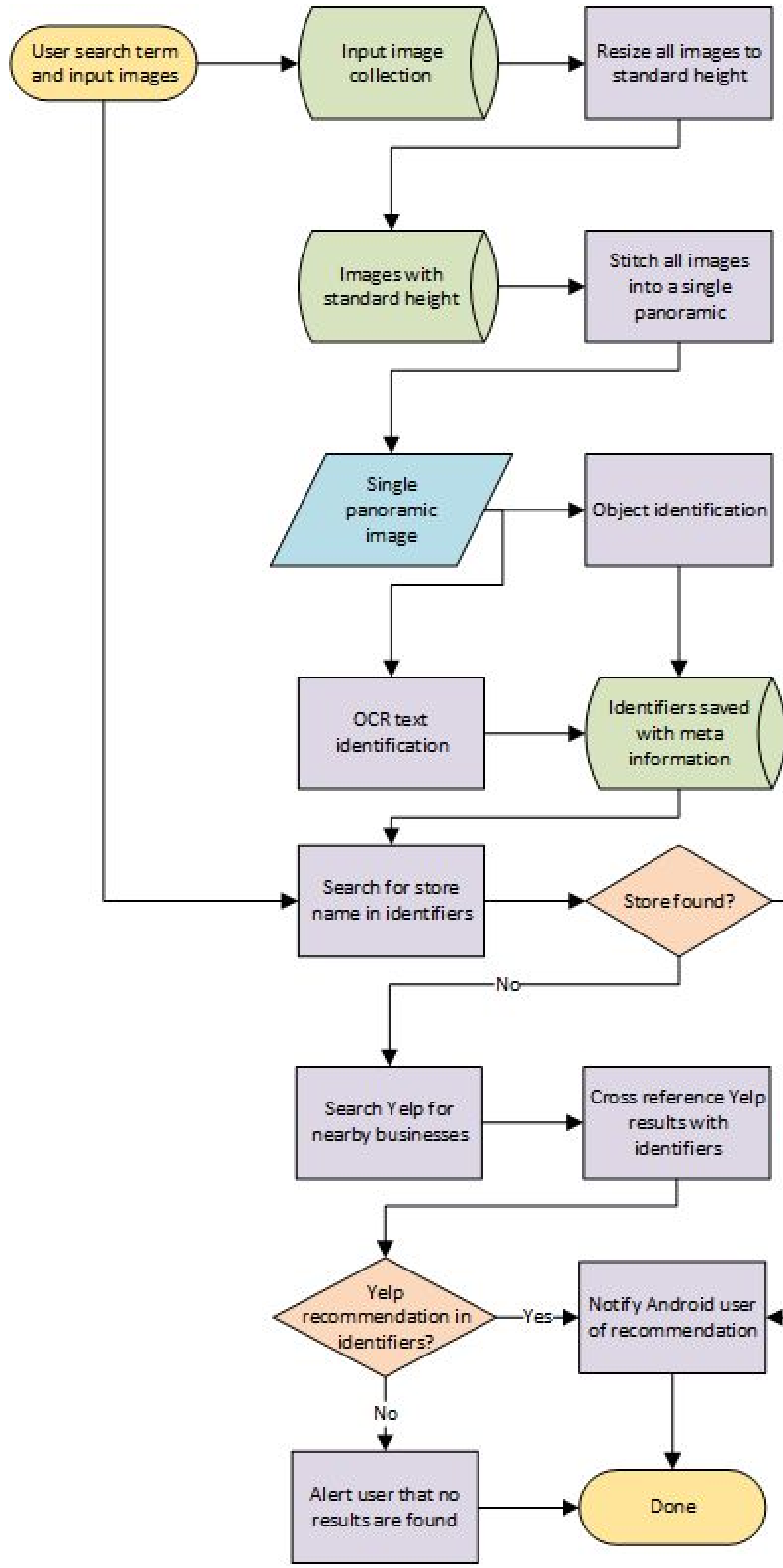
- Obstacle avoidance when the robot moves
- High success rate of accurate image detection
- Have the robot find the wearer of the smartwatch based on available methods
- Identify and use shortest path between points when the robot either finds the smartwatch or finds a previously identified image

Design

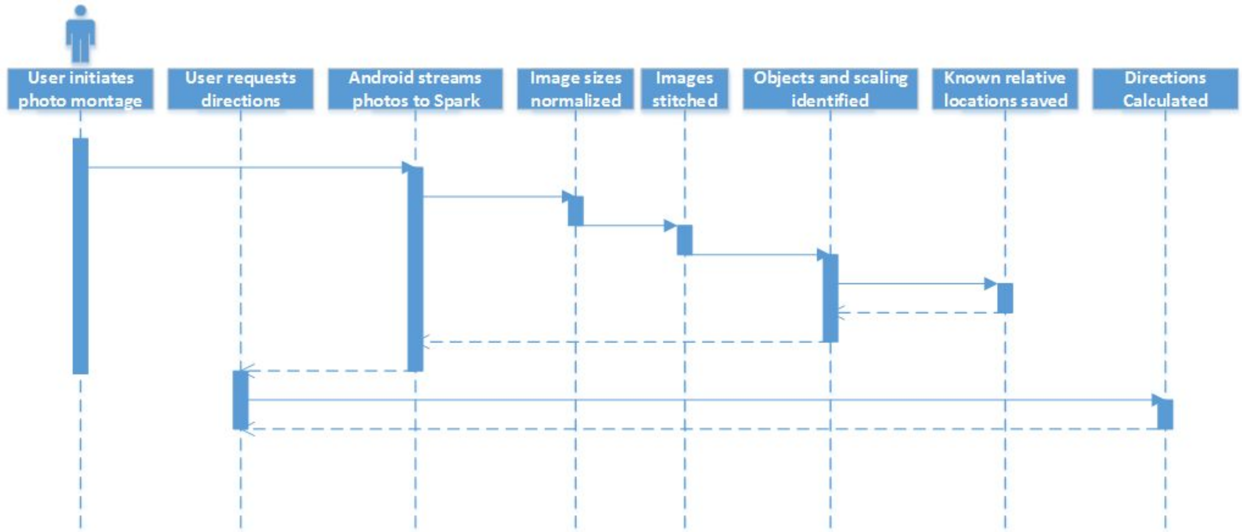
The design of the autonomous tour guide is based on four primary components. These are the user's smartwatch, the Android powered robot, the backend Tomcat server, and the backend Apache Spark cluster. The user's smartwatch is the primary user interface device, providing a speech to text capability and user notifications. The Android robot is responsible for physically navigating about the area, and sending a regular stream of images with geolocation information to the backend systems. The Tomcat server is responsible for keeping a record of the geolocation information received from the robot, and provides the "brains" for calculating movement commands to direct the robot. The Apache Spark cluster is responsible for image calculations, including keypoint identification and similarity scoring between images.



Basic design



Backend server process flow



High level sequence diagram

Individual components that have already been worked are described below.

Image matching engine

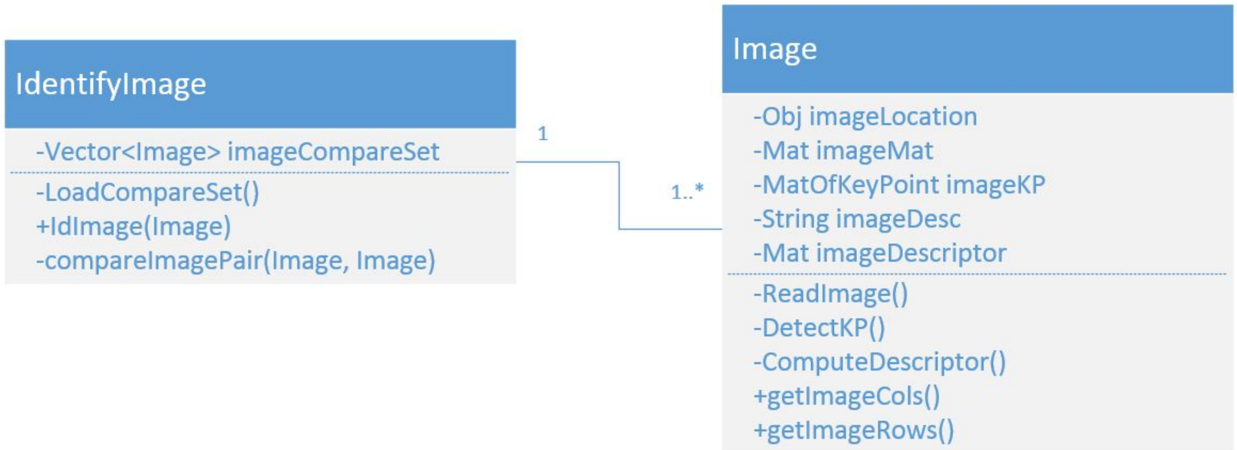
Source code repository:
https://github.com/smoeller1/BigData-Spring2016-TourGuide/tree/master/src/ImageId/src/ImageId

The image matching engine is written as Java classes utilizing libraries from the OpenCV project. The first class is the Image class, responsible for identifying and saving the key points for the image. The second class is the IdentifyImage class, responsible for comparing different Image's to each other.

Each individual image is saved as an Image object. Upon instantiation, the Image object reads in the image file using the appropriate method depending on the image source location. Next, the OpenCV FeatureDetector is used to identify the key feature points using the ORB method, though other methods can be utilized. Last, the OpenCV DescriptorExtractor extracts the descriptors of the image using the ORB method.

The IdentifyImage class automatically creates Image objects for each of the comparison images upon instantiation. The goal is to have the comparison images be self-learned based on a keyword list of images that should be learned. However, at this time the list of comparison images is hard coded into the LoadCompareSet method. When a new image needs to be identified, the calling program calls the IdImage method with the new image that needs to be identified. The IdImage method will compare this new image against the database of known comparison images to attempt to find a match. The image comparison is done using the

OpenCV Imgproc MatchTemplate to attempt to identify subsections of the new image, when possible.



Speech to text

Source code repository: https://github.com/SindhuReddyG-sgdd7/SpeechToText

The speech to text functionality utilizes the microphone on the smartwatch to allow the user to speak commands to the Android robot. The speech to text allows for text based commands to be easily processed by the Android robot.

REST Interface

Source code repository:
https://github.com/smoeller1/BigData-Spring2016-TourGuide/tree/master/src/ImageId/src/ImageId

The REST interface uses JSON formatted data to communicate between the Android robot and the backend servers. This interface is used for sending the images captured by the Android robot, as well as receiving messages from the servers when items are identified within the images, so that this information can be relayed on to the smartwatch.

Camera controller

Source code repository:
https://github.com/smoeller1/BigData-Spring2016-TourGuide/tree/master/src/ImageId/src/ImageId

The camera controller works on both a timer and a movement tracker. Each time the timer counts down, the camera controller will check to see if there has been movement since the last image was captured. If there has been movement, a new image is captured. If there has not been movement, then no image is captured and the timer is reset.

Statistical Analysis

The captured images are sent to the central server, where the image analysis is performed. The Images identified will be subjected for statistical analysis to infer insights from the obtained data. As part of image identification/recognition algorithm, multiple captured images are grouped together based on the content of the images and all the grouped images are given for the image comparator for identifying the correct object present within the image. For achieving this, statistical method of k-means is used to group the captured images. The working of the image grouping is tested based on assumptions which are expected to be completed for the final stages.

Sample Image data:

ImageName	Q1 saturation	Q2 saturation	Q3 saturation	Q4 saturation
Image0.png	0.3	0.5	0.6	0.2
Image1.png	0.1	0.6	0.2	0.5
Image2.png	1.4	0.0	1.3	1.4
Image3.png	1.1	2.0	0.4	2.6
Image4.png	3.4	2.1	2.0	0.9
Image5.png	4.2	4.4	1.0	3.1
Image6.png	3.1	0.4	4.1	4.3

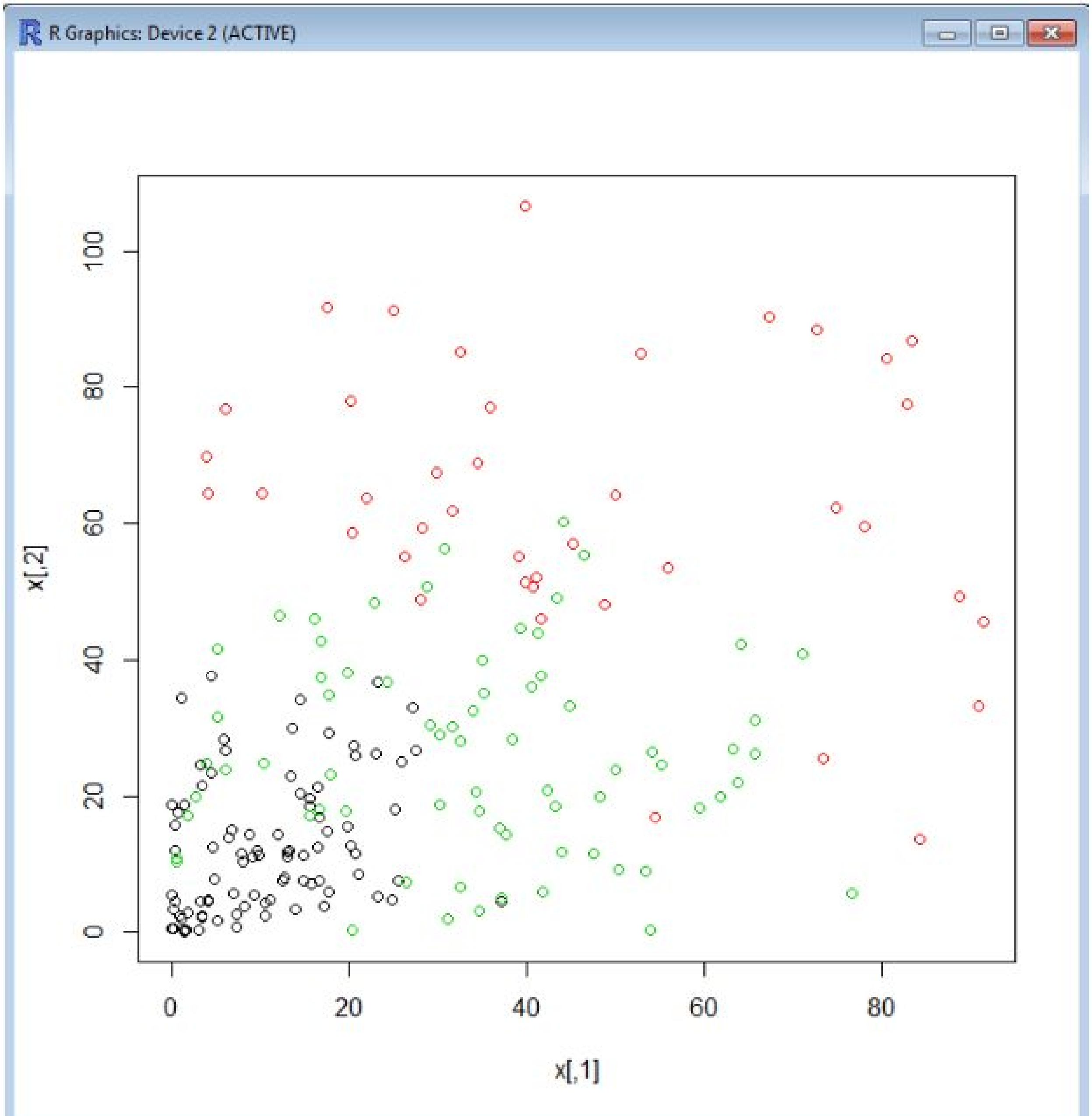


Figure 1: K-means clustering of images

Image normalization

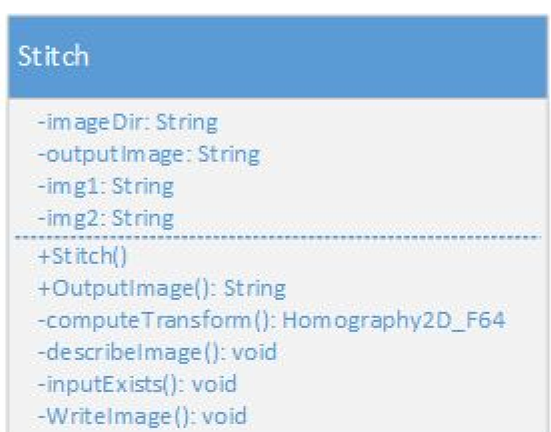
Source code repository: https://github.com/SindhuReddyG-sgdd7/ImageScaling

As the images captured will be of different sizes, it is required to modify the pixels of all captured images to the same pixel intensity and same size. An Abstract Window Toolkit (AWT), a JAVA built-in tool is used which renders the outline of any image. All captured images will be given as input and the normalized images are stored in a new folder which are then processed for image stitching.

Image stitching

There are three basic ways to achieve image stitching in Java. The first way is to write entire implementation from scratch, implementing feature identification, feature matching, rotation, and scaling. This method, while a great learning experience, would take more time than is available in a single semester course. The second method would be to use the OpenCV C++ libraries, and compile them through a Java interface using SWIG or JavaCPP. This method is feasible, but does introduce added complexity to the code maintenance. The third method, which was chosen for this project, is to use the BoofCV libraries. These libraries provide some of the same features as OpenCV, but are written as native Java code.

When stitching a large number of images into a panoramic, the Stitching class can be parallelized in a binary tree fashion. On the first pass pairs of consecutive images will be stitched, with all pairs stitched concurrently (as far as hardware will allow). After the first pass there will be half as many images, which will be paired up again and stitched concurrently. This will continue until there is a single image left. The number of iterative parallel calls to Stitch would then be log(N).



Class diagram for Stitch class

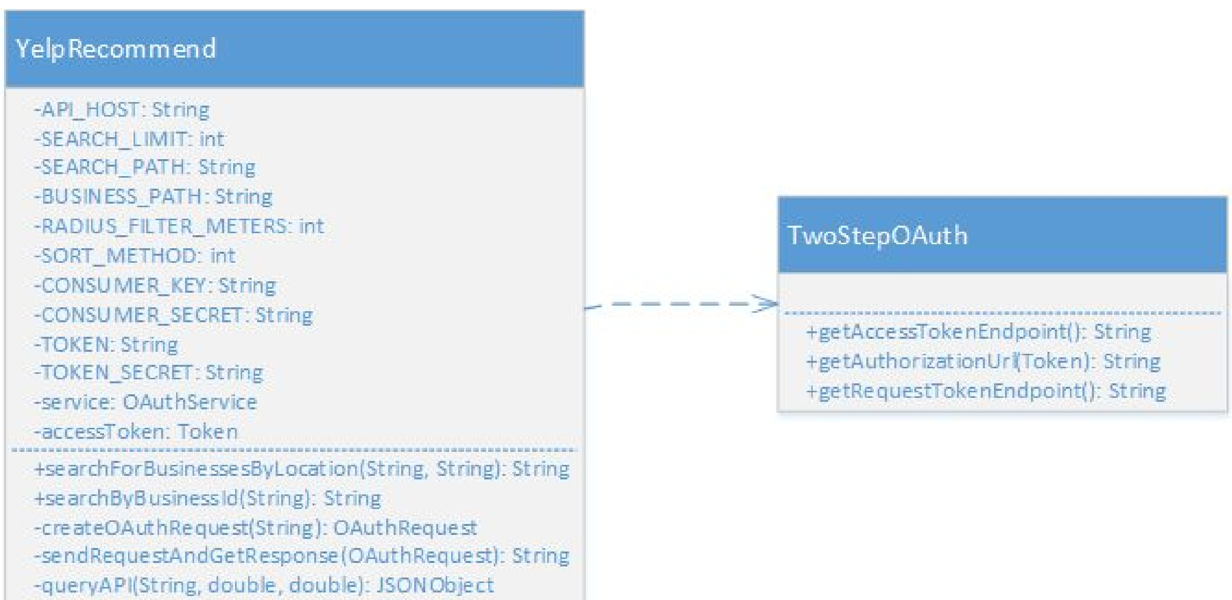
The underlying algorithm that is used to identify the descriptors is the Speeded Up Robust Feature (SURF) algorithm. This algorithm is similar to the SIFT algorithm, but SURF is considered to be faster. The SURF algorithm has three main steps, interest point detection, local neighborhood description, and matching.

There is an outstanding defect with the image stitching class which is causing the stitched image to be created within a large, black image of static size. This will need to be addressed to have the stitched image size be based on the actual dimensions of the stitching, and have any additional space coded as white instead of black. This will also be useful when stitching images that have previously been stitched.

Item Searching

Github location: https://github.com/smoeller1/BigData-Spring2016-TourGuide

Once an area is mapped, a user will want to be able to search for the best nearby business that can meet the shopping needs of the user. This nearby business searching is being provided by Yelp through the Yelp Developer API Version 2. This is a REST interface into the yelp system. By default the single highest rated nearby business that provides the item or service the user requested will be returned.



YelpRecommend class diagram

The Yelp API is accessed by OAuth v1.0a, with a set of authorization keys generated just for the Tour Guide application. By default the YelpRecommend class will restrict the search area to within 500 meters of the user, and will return the single highest rated business that meets the user's search criteria. According to Yelp, "highest rated" is based on an algorithm that incorporates both average user rating and number of ratings, in order to prevent a business with a single 5 star rating from dominating the returned results.

Sample Java usage:

```
JSONObject results = queryAPI((String) "Jeans", 38.952508, -94.719309);
System.out.println("Best match business: " + results.toString());
```

Results:

Best match business:
{"coordinate":{"latitude":"38.95641","longitude":"-94.71533"},"name":"Nordstrom"}

Text Recognition

Git link: https://github.com/sudhakarCS5542/RoboGuide.git

Text recognition is another processing part of the image section to extract the text present in the images such as brand text or to read signboards like exit or restrooms. The text present in the images is read using a library called Tesseract. Tesseract is an open-source engine developed initially by HP, later improved heavily by community groups. Using the Tesseract OCR engine, we are working on text recognition algorithm to detect English language based text present in the captured images. The Tesseract OCR engine has three different libraries which focus on the accuracy of the text present in the images. Each library works on user mentioned time to work on processing the images.

The images captured in the smartphone camera are pre-processed to Bitmap images, which the OCR engine can work on. Before the images are fed to the OCR engine, any skewness present in the images is removed and also the images are rotated to get appropriate orientation that Tesseract can work. Along with removing the skewness, the images are converted with ideal parameters for achieving better results (dpi - 300, black text on white background).

External APIs

OpenCV - C++ and Java Image processing libraries. http://opencv.org

BoofCV - Java Image processing libraries. http://boofcv.org

Yelp API - Reviews and rankings of nearby businesses.
https://www.yelp.com/developers/documentation/v2/overview

Project Plan

The project is divided into four iterations. The iterations are described in more detail below. In general, with each sprint each team member is focusing on finishing a single feature. At the start of each sprint the remaining features are divided among the team members, with all team members working on all parts of the system over the course of the project.

Iteration 1

The four features that were worked during iteration 1 were speech to text, REST communications interface, camera controls, and image identification. These four features are described in more detail above. The work this sprint was completed by the following team members:

Sindhu Golconda - Speech to text
Simon Moeller - Image identification
Prudhvi Mudunuri - Camera controller
Sudhakar Peddinti - REST communications interface

We attempted to use ZenHub to help track work, however ZenHub went down (or just stopped working) for all team members partway through the iteration. Since the team members already understood what needed to be done, we continued to work without ZenHub.

All team members communicated well during the iteration, and the team met in person multiple times over the course of the iteration. This good communication also helped the team to largely achieve the goals of the iteration. There are a few minor defects still being resolved in some of the code from this iteration, but it is anticipated that these debugging efforts will be complete by this weekend.

Iteration 2

The focus of iteration 2, based on feedback after iteration 1, was altered to focus more on image detection and manipulation processes. As such, new features were added such as optical character recognition, image normalization, and panoramic image stitching. Also based on the feedback from iteration 1, some of the original features have been lowered in priority. These include robot movement, some of the smartwatch interactions, and other features unrelated to direct image processing.

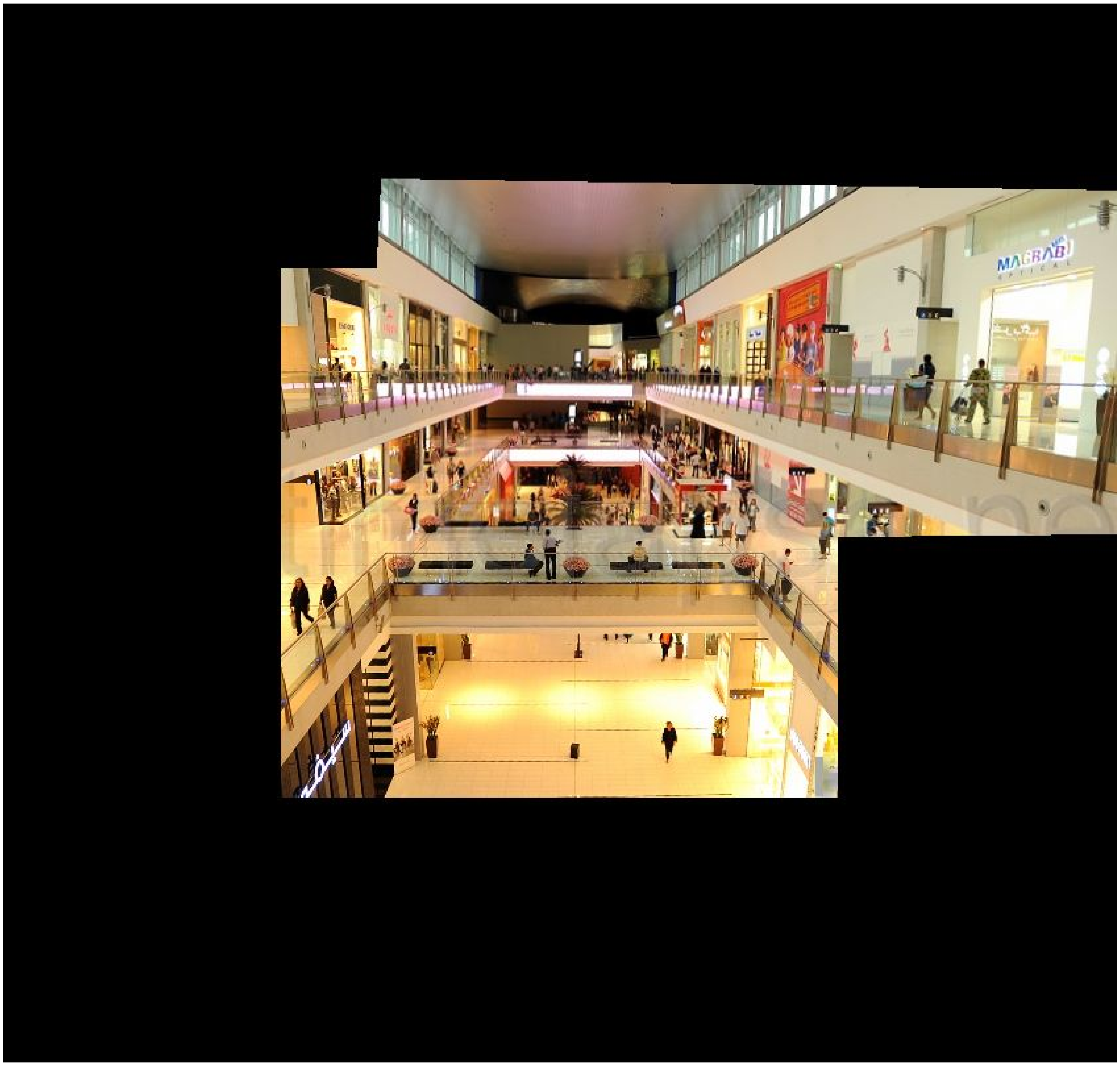
Sindhu Golconda - Image normalization
Simon Moeller - Image stitching
Prudhvi Mudunuri - Optical character recognition
Sudhakar Peddinti - Optical character recognition

The team met two to three times each week during iteration 2. The meetings were generally treating as status and design discussion sessions. After the change of direction for the project following the iteration 1 feedback, it did take a few team meetings before the new scope of the project was finalized. This meant that some additional time this iteration was spent on basic research tasks to identify what the new project scope could realistically be, and what the new priority of the features should be.

Example of image stitching

Input Image 1

Input image 2 (image scale is 0.5 of Input image 1)



Output stitched image

Iteration 3

Based on the feedback provided on the Iteration 2, we have focused on the Image processing and the recommendation system for this iteration. In this phase we have implemented the Image classification system to categorize the captured images to identify the object such as Direction boards, Exit boards, store names. The classification system uses Random Forest algorithm to classify images into the different categories. We have used few Exit sign boards images at different angles, different types to train the classifier. Also we have implemented a recommendation system using Yelp data to provide recommendation to the user based on the request that user provides. The recommendation is made to use the ratings and reviews that the user previously provided. We have plan to implement sentiment analysis on the reviews so that we can make better recommendations according to the likes and dislikes of the user. For the current iteration, we have implemented the system to read the user data and provide a store with the name and geo location details to the user's smartphone.

Sindhu Golconda - Distance estimation from an object
Simon Moeller - Business recommendations
Prudhvi Mudunuri - Object classification
Sudhakar Peddinti - Object classification

Sample recommendation: The user wishes to find a store that sells jeans near the current location. The current location is provided as GPS coordinates from the user's smartphone.

```
JSONObject results = queryAPI((String) "Jeans", 38.952508, -94.719309);
```

Results:

Nordstrom, located at 38.95641, -94.71533

Iteration 4

Iteration 4 has been utilized to integrate all of the development efforts from the previous iterations into a single product. This effort has introduced many additional issues to debug, due to slightly different expected interfaces and different, incompatible library dependencies. For example, one image manipulation class was written using the BoofCV 0.23 libraries, and another was written using BoofCV 0.22 libraries. The change from 0.22 to 0.23 represents a very major change in this library, including a renaming and refactoring of a large number of classes and methods within the library. None of the integration challenges have proven to be insurmountable, though.

The Autonomous Tour Guide, as it is now, is comprised of a subset of the original goals of the project. The original project use case envisioned an autonomous robot that would tour a location ahead of and independent from a human operator. The robot would capture a stream of images which would then be fed into the backend image processing for identification of and location tracking for any identifiable objects. After a period of time, the human operator would then be able to ask a smartwatch for directions to a desired search term, for example a store name or a location like “nearest exit”. The backend would process this search term against all identified objects, and if found direct the user to the searched object via relative directions.

The final product requires a series of images to be given to the processing engine manually, without the use of a robot. Additionally, the implementation could be described as “rough around the edges”, as the user interface is very rudimentary, and some desired image processing features have proven to be more complex than expected to implement with a high level of accuracy.

All four team members worked together on the integration of the various features.

Bibliography

OpenCV libraries. http://opencv.org

BoofCV libraries. http://boofcv.org

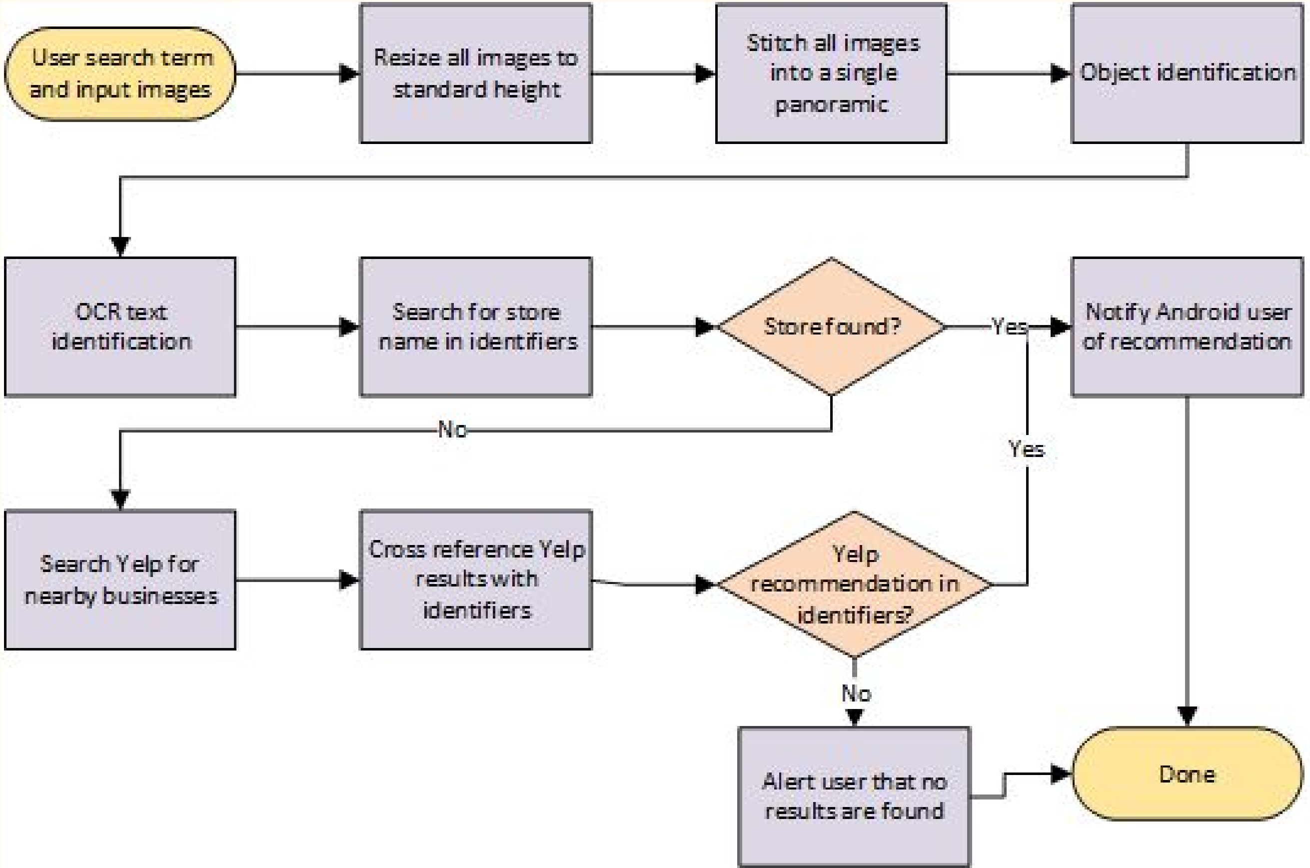
Yelp API V2. https://www.yelp.com/developers/

Tesseract OCR libraries: http://tess4j.sourceforge.net

Autonomous Tour Guide

Team 6
Sindhu Reddy Golconda
Simon Moeller
Prudhvi Raj Mudunuri
Sudhakar Reddy Peddinti
https://github.com/smoeller1/BigData-Spring2016-TourGuide

Server HLD



User search term and input images
Resize all images to standard height
Stitch all images into a single panoramic
Object identification
OCR text identification
Search for store name in identifiers
Store found?
Yes
Notify Android user of recommendation
No
Yes
Search Yelp for nearby businesses
Cross reference Yelp results with identifiers
Yelp recommendation in identifiers?
No
Alert user that no results are found
Done

Stitch

1. Run a SURF detection against this image and previous
2. Generate a similarity score between the features in the two images uses Euclidian distance
3. Use a RANSAC algorithm to identify the best matched features to use to stitch the images
4. Create a new image of size 2*(previous image size), and draw the matched, stitched image

Input Data:

First iteration - all image files found in the input directory

All iterations after first - an ArrayList of all sub-panoramic images from the previous iteration is passed

Image Stitching size verse time
1800
1600
1400
1200
1000
800
600
400
200
0
18
16
14
12
10
8
6
4
2
0
1
2
3
4
5
6
7
8
9
y = 1.8917x - 1.5106
image sizes (pixels)
time (s)
Linear (time (s))

IdentifyImage

1. ORB is used for feature detection, due to performance (2 orders of magnitude faster than SIFT with similar quality of results)*
2. The normalized correlation coefficient method is used for matching the objects in the images
3. Random Forest model used for training

Data:

Training/testing data: Static, labeled training data set collected from Google searches

Using two categories, Exits and Bathrooms, with 32 images, achieved an image detection accuracy of 87.5%

Image OCR

Utilizes the tess4j Tesseract libraries

Tesseract libraries provide optical character recognition for whole images (as utilized here), or for subsections of images

Data:

Training data: static data collected from the tess4j SourceForge site

Testing against 33 images:

- 9.1% successfully read entire keyword
- 27.3% successfully read part of keyword

Credits

This work was done in partial fulfillment of the requirements of CS5542: Big Data Analytics and Apps, CSEE Department, University of Missouri–Kansas City (Spring 2016). Instructor: Dr. Yugyung Lee, TAs: Mayanka Chandrashekar, Feichen Shen.