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.

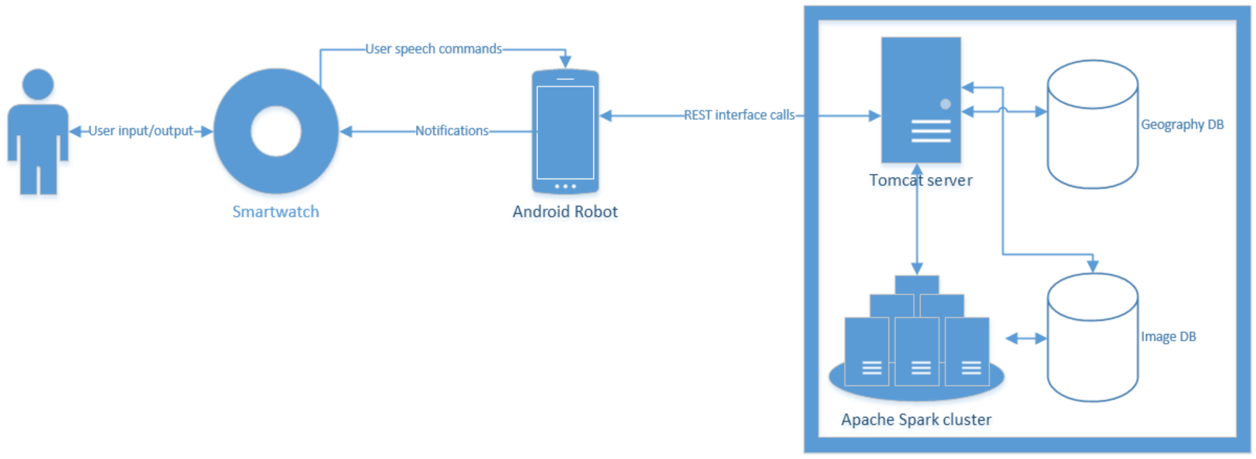
Project goals:

- 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

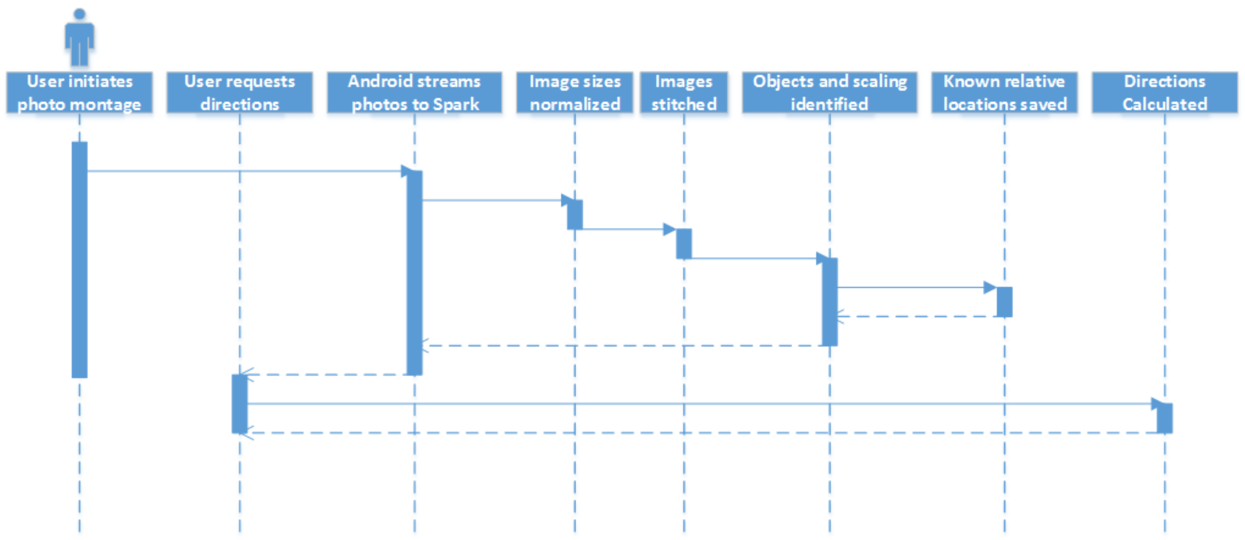
    - Identify and use shortest path between points when the robot either finds the smartwatch or finds a previously identified image

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


User initiates photo montage
User requests directions
Android streams photos to Spark
Image sizes normalized
Images stitched
Objects and scaling identified
Known relative locations saved
Directions Calculated

High level sequence diagram

Individual components that have already been worked are described below.

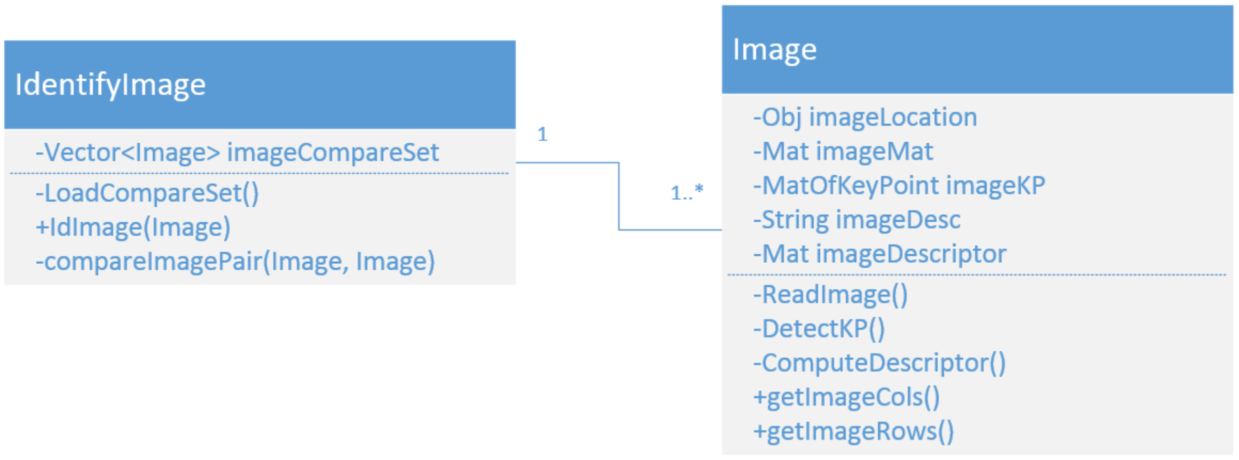
Image matching engine

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/sudhakarCS5542/RoboGuide.git

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

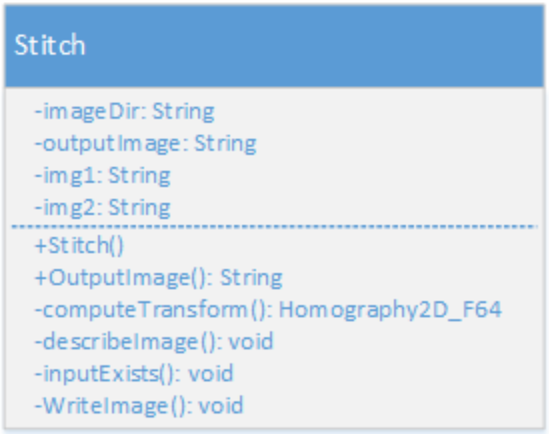
Source code repository: https://github.com/smoeller1/BigData-Spring2016-TourGuide/tree/master/src/ImageId/src/ImageId

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 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).



```
Stitch
-imageDir: String
-outputImage: String
-img1: String
-img2: String
-------------------------------------
+Stitch()
+OutputImage(): String
-computeTransform(): Homography2D_F64
-describeImage(): void
-inputExists(): void
-WriteImage(): void
```

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 - Depth detection
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.

Image normalization

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

Optical Character Recognition:
Using the opensource Tesseract OCR engine, we are working on text recognition algorithm to detect English language based text present in the captured images. The Tesseract OCR engie 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.

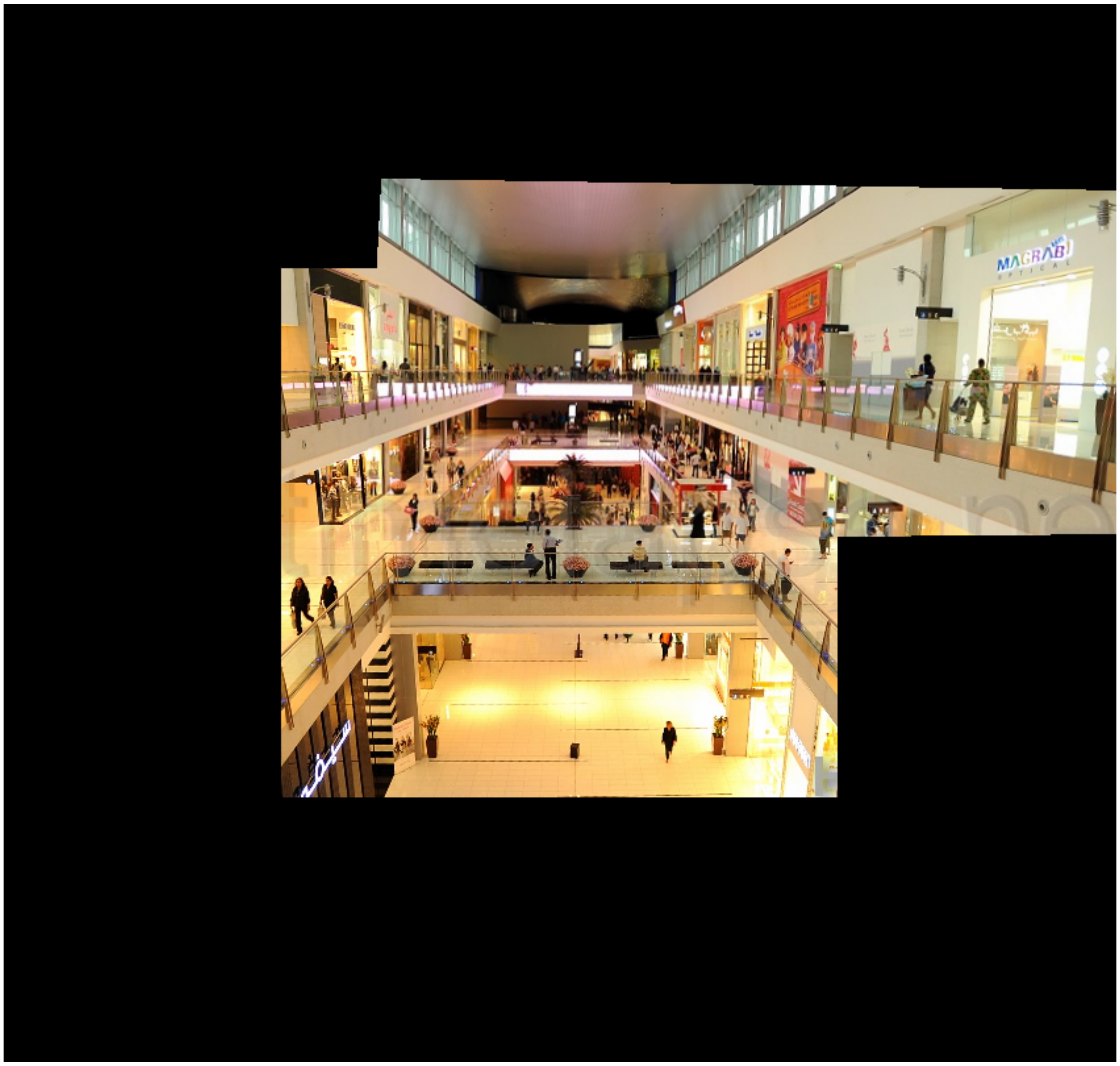
Source code repository: https://github.com/sudhakarCS5542/RoboGuide/tree/master/src

Edge Detection:
Depth detection is being implemented by using the concept of edge detection, where in each image is subjected to triangulation method, where in the pixels of the image object are represented in the form of matrix and it is transformed using canny edge technique. Depth is calculated, by comparing the number of pixels with respect to the traced object to decide whether the object is nearer or far with respect to the camera which inturn to the robot.
Source code repository: https://github.com/prudhvirajmudunuri/ProjectRoboMe.git

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