Design Document

Team: Smart City

Date: 28 April 2018

1 Table of Contents

2 Revision History

Date	Author	Revisions Made
02/16/17	Eric Jin Wook Choi: SP17 Project Manager	• Design Document Creation • Project Charter • Semester Information
04/14/17	Eric Jin Wook Choi	• Current Design
9/27/2017	Wesley Sawyer, Grant Hilbert: FA17 Hardware Team	• Fall 2017 End-Semester update
10/6/17	Eric Jin Wook Choi: FA17 Project Manager	• Fall 2017 End-Semester update
11/29/2017	Eric Jin Wook Choi	• Fall 2017 End-Semester Update
11/30/2017	Hardi Sura	• Fall 2017 End-Semester Update
02/16/2018	Erika Lai Ting Lin: SP18 Project Manager	• Project Motivation • Addition of Links • Outcomes/Deliverables • Semester Information
04/05/2018	Erika Lai Ting Lin	• Reorganization of Design Document
04/19/2018	Erika Lai Ting Lin	• Final Design Review Comments Reflections
4/28/2018	Erika Lai Ting Lin Romita Biswas Kalpan Jasani Kartik Mittal Ayuub Jose	• End of Semester Updates • Contour Tracking

3 Design Status

Phase 1: Project Identification	**Status:** *Completed* **Semester:** *Spring 2017*
Phase 2: Specification Development	**Status:** *Completed* **Semester:** *Spring 2017*
Phase 3: Conceptual Design	**Status:** *Completed* **Semester:** *Spring 2017*
Phase 4: Detailed Design	**Status:** *In Progress* **Semester:** *Spring 2018*
Phase 5: Delivery	**Status:** *To be done** **Semester:**
Phase 6: Service / Maintenance	**Status:** *To be done** **Semester:**

4 Project Charter

4.1 Description of the Community Partner

The project partner for the Purdue Engineering Projects In Community Service (EPICS) Smart Cities Team is the City of West Lafayette Department of Engineering ("The City"), whose primary responsibilities are:

1.) to evaluate engineering solutions.
2.) to provide recommendations to the Mayor, other city departments, boards, and commissions regarding private and public works throughout the city[1].

The City has requested EPICS to develop an integrated system that will detect, observe, analyze, and suggest fixture recommendations for "potholes" within City limits. Potholes, are road defects where road materials erode and form holes due to erosion from weather and vehicle travel rates, etc.[2].

Potholes may oftentimes result in harm to the community ranging from pedestrians to vehicle operators by causing vehicle damages as well as putting lives at risk. "Out of 33,000 traffic fatalities per year, one-third involve poor road conditions"[3] and United States (U.S.) motorists spend approximately "$67 billion a year in extra for repairs and operating costs due to the poor conditions of roads."[4] To combat increasing frequency and severity of potholes, the City has requested EPICS to tackle the problem through Smart City design.

The City has expressed concerns regarding ongoing City efforts to provide pothole fixtures within City limits; visual inspection is the primary source of detecting potholes through Pavement Surface Evaluation and Rating (PASER).[5] Marcus Smith, assistant city engineer and Smart City's primary contact with The City, provided general comments regarding The City's needs and user/stakeholder identification.[6] The City is seeking to improve its processes with pothole detection and location tracking. The currently implemented PASER and fixture schedules through the Department of Street & Sanitation require manual labor and do not always produce accurate results. The project partner's overall mission is to increase the efficiency of the road damage identification system within the city of West Lafayette. Smart City's end-deliverable is to develop an autonomous system that satisfies the City's needs and will be delivered to the Department of Engineering for implementation.

As a result of this design, the Department of Engineering will be able to recommend appropriate adjustments to the Mayor's office and other city departments in hopes of making the pothole detection process more effective through a cost efficient and reliable design. Ultimately, the project partner will be able to effectually detect potholes, thus being able to effectively repair road damage, making roads safer for the West Lafayette community.

---

[1] http://www.westlafayette.in.gov/engineering/
[2] https://www.fhwa.dot.gov/pavement/pub_details.cfm?id=139
[3] https://www.pothole.info/the-facts/
[4] http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=7556304
[5] http://www.in.gov/indot/2469.htm
[6] https://sharepoint.ecn.purdue.edu/epics/teams/smartcity/_layouts/15/WopiFrame.aspx?sourcedoc=/epics/teams/smartcity/Semester%20Documentation/Spring%202018/Project%20Partner%20Information/PP%20Questions%20and%20Answers.docx&action=default

4.2 Stakeholders

The stakeholders of this project include those who will be affected other than the project partner and those who have interest in the project's success. As this project focuses on a design specifically for the City, the primary user is the West Lafayette Department of Engineering. However, this pothole detection system will impact others as well:

- Primary Users
  - The City of West Lafayette Department of Engineering
    - 3-5 city engineers with respective sub-department interns.
      Provision: A platform that will help identify/located potholes such that the city is able to communicate a fixture schedule to the Department of Streets & Sanitation (secondary users).
  - West Lafayette Community/General Public
    - Provision: An application allowing the community to report road conditions throughout the city, i.e. pothole detection by public reachability.
- Secondary Users
  - Other departments and employee, eg. Streets & Sanitation, Internal, etc.
- Primary Stakeholders:
  - The City of West Lafayette Department of Engineering
  - Purdue University EPICS
  - West Lafayette Community and General Public

4.3 Project Objectives

4.3.1 Project Operations & Logistics

The Smart City EPICS Team is split into three sub-teams: Hardware (HD), Data Analysis (DA), and Website and Application Development (WA) - all three teams need to work collaboratively to achieve the overall goal. Each sub-team consists of 3-4 members directed by a Design Lead. The motivation for this project is the lack of an efficient and autonomous method of pothole detection in the city of West Lafayette, thus resulting in increased risk for the community. The HD Team will create a sensor-analysis system that can be attached to existing city public service vehicles, and the DA Team will analyze the gathered data to decipher the presence and severity of potholes. The system will eventually be implemented on city vehicles: specifically for the Spring 2018 semester, the team will focus on implementation on garbage trucks. The WA Team will develop a website for city use to view pothole severity and location while also producing a smart-phone application accessible to the general public such that they are able to directly report potholes and issues throughout the city. The mission of the project partner is "to develop a vehicle mounted scanner to detect potholes and report their size and location" and "to develop an application for residents to submit issues to the city."[7] Smart City's design aims to satisfy the mission of the project partner while also achieving individual team goals.

Each team has developed an independent design process according to their need-finding and project deliverable goals and specifications. In the overall scope, individual design processes merge towards the end-deliverable, but team dynamic, management, and procedure should retain individuality. The "EPICS

[7] https://sharepoint.ecn.purdue.edu/epics/teams/smartcity/_layouts/15/WopiFrame.aspx?sourcedoc=/epics/teams/smartcity/Semester%20Documentation/Spring%202018/Project%20Partner%20Information/PP%20Questions%20and%20Answers.docx&action=default

Smart City Project Design Document" is compiled to illustrate the progress and design process across the whole team. The Design Document will encompass *all* of the work done in previous semesters and current semester, displaying the entire design process that has been iterated.

4.3.2 Project Motivation

Potholes present a high risk for the community. The current approach the City of West Lafayette utilizes, described below, is not integrated nor automated and requires much human effort/interaction. Pothole repairs are unable to be completed all at once:

1. Vehicle and personal damage arising from poor road conditions
2. Need for identifying pothole location: GPS coordinates; streets with most potholes
3. Determine pothole severity
    a. Currently using PASER - not optimized and requires much human-interaction
    b. Visual inspection by City interns who "rate" the pothole severity by PASER standards of differing road conditions - inefficient and new damage may evolve within reporting-fixing time frame
4. Make repair recommendations
    a. Engineering department recommends to Mayor's office
    b. Fixtures completed by Street & Sanitation department
    c. Repair crews are sent to the most critical sections to patch and address any and all damage

The City currently does "patchwork" fixture for potholes, i.e. filling potholes until a certain level is reached where the pothole can no longer be ignored. The City also does patchwork for an entire street until the total threshold of potholes on a given street cannot be neglected. We hope that our system can locate and archive pothole fixtures so that The City can have an accurate assessment of pothole thresholding.

Other pothole solution options are offered by private companies such as Google, Automatic Road Analyzer (ARAN), Land Rover, and Ford:

Product	**Details**
Google Patented Pothole Detection System	Developed for later implementation on Google self-driving cars and is not readily available[8]
ARAN	$1.5 Million military-grade vehicle that operates once every two years[9]
Land Rover and Ford	Motor vehicles that detect potholes to allow the vehicle to adjust using a pneumatic system with the intention of decreasing vehicle damages[10,11]
EPICS Smart City Design	Cost effective yet efficient autonomous pothole detection system that is able to locate and quantify potholes on a regular basis.

[8] US Patent 9,108,640 B2
[9] http://pavement.com.au/equipment/automated-road-analyser/
[10] http://www.landrover.com/experiences/news/pothole-detection.html
[11] https://media.ford.com/content/fordmedia/fna/us/en/news/2016/02/18/all-new-ford-fusion-v6-sport-helps-protect-against-potholes.html

The table, shown above, displays an overview of products that achieve similar objectives of detecting potholes. The last row is our product and how it differs.

These are solutions that are directed in a similar direction, but there is no one existing solution that is cost efficient, effective, and autonomously locates and quantifies potholes. Our solution is an efficient, simple, attachable sensor that can be implemented on existing City public service vehicles (garbage, police, administrative, etc.).

4.3.3 Project Specification

Based on initial need-finding and background identification of users/stakeholders, Smart City has determined the following requirements and limitations for the project:

- **Objectives or goals:** The design will be cost-effective yet efficient and autonomous, that the City is able to utilize with city vehicles with the goal of locating and quantifying potholes. The system's results will be accessible to the city via a website, and an application will be accessible to the public to map potholes as well as report road issues to the city.
- **Constraints:** The design must be cost efficient yet produce an effective analysis as potholes throughout the city are frequently autonomously monitored. The design must also be easily accessible and adaptable, and the system cannot be drilled into city property for attachment.
- **Function**: A sensor will capture continuous frames of the road to collect data with respect to GPS location. The data will then be analyzed to determine whether a pothole exists as well as its severity. A website and application will be developed to visualize the location and severity of potholes, and for the general public to report additional road issues.
- **Implementations or means:** The design will be implemented on city vehicles that frequent the roads in this city. Specifically for the Spring 2018 semester, the design will be focused on the implementation with city garbage trucks.

Smart City acknowledges that primary users have a mission of an easily operable and autonomous system, thus concluding that the end-deliverable should depend on minimal technical specifications and should be easily usable for the appropriate audience.

4.3.4 Proposed Solution

Smart City proposes a solution to develop a sensor system that can be easily attached to vehicles that frequent city roads that is cost efficient yet effective. Based on a profile of The City's needs, Smart City's end-deliverable should be able to provide visualization and digital documentation of potholes' location and severity and be able to communicate with appropriate technology and users/stakeholders for completeness.

The HD Team is designing a system that will capture images and location information, and the DA Team will analyze the data to deliver the severity of potholes at given locations. The WA Team will provide an end-user functionality by developing a website for city use, allowing the relevant authority to access location and severity of potholes. The WA Team will also be creating a downloadable application that will allow the City to receive feedback from the public regarding city issues while also alerting the public about location and severity and potholes.

4.4 Outcomes/Deliverables

The HD Team and DA Team work closely to develop the hardware and software into a product that will capture images necessary for pothole analysis, constituting identification, classification, and fixture recommendations. The WA Team then takes the analyzed data and displays the location and severity of potholes on a map.

4.4.1 Hardware Team

Whereas Spring 2017 and Fall 2017 combined the hardware and data analysis team, Spring 2018 decided that with fewer team members, it is more practical to allocate separate sub-teams since all three sub-teams - HD, DA, and DA - need to work laterally. For Spring 2018, the HD would like to accomplish two functional tasks: modified Kinect-imaging system and GPS location services. The first milestone is to design an *Initial Prototype* that will be able to execute indoor testing in which the Kinect sensor, a line of motion sensing input devices that was produced by Microsoft for Xbox 360 and Xbox One video game consoles and Microsoft Windows PCs,[12] will detect and record functional tests - artificial potholes distanced from a car bump and road. The initial prototype will measure depth heights - distance from sensor-bump and road. The initial prototype is intended to be "rough" prototype that will be a basic model that can be fine-tuned. The second milestone is to develop a *Functional Prototype* that can be used for Functional Prototyping. The functional prototype will be field tested on appropriate pothole-roads to insure the quality of our GPS and pothole depth data. In the scope of design schedule, the team plans to develop at least a functional prototype and may need further functional evaluation and *Revised Prototyping & Field Testing* due to technology modification and time constraints.

4.4.2 Data Analysis Team

The DA Team is responsible for developing the technology for analysis that would determine conclusions and recommendations. This would involve identification and classification of potholes, from the data gathered from the sensor. DA intends to develop an integrated system that can help The City timely receive pothole information with the GPS location and severity "level" (Low-Medium-High) of each pothole correlates.

The main deliverable of Spring 2018 EPICS Smart City is to automate the process of detecting and quantifying the severity of potholes:

- Using the Microsoft Kinect V2 (Kinect), (red-green-blue (RGB) depth values) needed to evaluate road condition
    - Kinect: image-processing tool used to capture photos (takes pictures of potholes and provides RGB depth data)
- Evaluating the accuracy of these sensors in measuring depths
- Coordinating the data collected from the Kinect and ultrasonic sensor with that from GPS to identify the location of the road surface defect
- Identifying and isolating a pothole from the database of all collected data
- Quantifying the severity of an identified pothole based on appropriate guidelines
- Creating a user-friendly GUI for end users to review the processed data
- Implement addition features to assist the city in road damage reporting
- Design and construct an arm to mount the K2 to a vehicle

Note: Past semesters attempted to implement ultrasonic sensor to complement the Kinect, but Fall 2017 opted out of this proposal (ultrasonic sensor receives only one value when the team needs "entire" values of a pothole).

Like Spring 2017 and Fall 2017, Spring 2018 the HD Team and DA Team continues to take on the responsibility of developing the analysis technology that would determine conclusions/recommendations - identification, classification, and fixture recommendations - from data measured by sensors (Ultrasonic, Kinect, and GPS). DA also continues to develop an integrated system to assist the city in simultaneously determining severity and location of potholes . For Spring 2018, the HD and DA Teams aim to execute

[12] https://msdn.microsoft.com/en-us/library/hh438998.aspx

what was intended in previous semesters. Specifically, the teams hope to organize previously written code and execute the sensor with accurate GPS location services while also formatting data in such a way that is able to be efficiently analyzed by future Smart City teams.

4.4.2 Website and Application Team

For Spring 2018, WA plans to provide an application featuring front/back-end graphical-user-interface (GUI) that displays a map of The City with locations and severity level of detected potholes. Additional functionality will depend on user preference and time-constraints. A website will be accessible to the city and an application will be accessible to the public. Smart City City Pothole Application is an "additional" functionality for the Smart City end-deliverable. The City will receive data - written/verbal complaints, photo responses, etc.- from citizens regarding potholes and road issues throughout the city. For Spring 2018, the WA Team would like to provide the project partner with a smart-device application as an interface to analyze citizen-feedback as well as locate and track potholes. The overall goal is for direct users, citizens and general public, to be able to download the end-product through the Apple Application Store and Google Play Store in order to deliver feedback through an application over directly contacting The City. Administrative users, city engineers, road maintenance, etc., will be able to access a map of road damage throughout city limits. In scope of the design schedule, WA plans to provide a *Conceptual Design & Initial Prototype* that provides the framework for future adjustment.

The result of this project aims to be an application for both iOS and Android mobile devices. This application will allow its users to report the location of a pothole or any other road damage, thus alerting the city to the necessity of a repair. The city will be looking out for the most frequent areas that are reported, as well as the most severe potholes that need to be repaired. This will allow them to access a map on which the damage reports will be compiled. However, general users will not be able to see this map. This allows repetitive submissions, and helps show how relevant and popular the road is in the community. Additionally, users will be able to submit photos and descriptions of potholes observed. This will allow the city to determine the severity of each pothole and establish whether or not the pothole needs immediate attention.

The main project goal is to design a smartphone application that will allow users to submit road damage reports in the City of West Lafayette. We hope to provide our project partner with an application that meets all needs and offers the City of West Lafayette in an efficient way to make roads safer. The purpose of this project is to assist city officials in locating potholes and other road damages by compiling the submitted information and projecting this data onto a map of the city. By providing this interface to the department of Road Safety in West Lafayette, it will be easy to locate and repair road damage. The WA Team will provide the public with a platform on which they can identify and report poor road conditions in addition to potholes. This capitalizes on all the pedestrians that travel across the town along the streets and sidewalks. While the application users travel through the city, they are likely to observe the road. It is our hope that they will take the time to document and report all poor road conditions that they identify during their travels. This data will be sent to the DA Team to be compiled with the data collected from the HD Team, thus allowing the resulting analysis to be displayed through the application by the WA Team. This compilation of data will be presented to the project partner, thus providing a way to identify road damage location throughout the city, the severity of each incident, and how often specific damage is reported.

The users will be able to download the application to their mobile phone free of charge, take a photo of the pothole, add a description, and submit this information in a common form. The data submitted from the various users will be mapped onto an outline of the city. This interface will be available to the application's administrator. Through accessing this map of the road damage throughout the city, the Road

Safety department will be able to address the reported damage. The map will be accessible through means of a website (HTML).

At the end of Fall 2017, the team left a basic functioning application as well as the code to design the application. This will allow any evolving needs to be addressed through adaptation and updates of the application.

Spring 2018 continued where Fall 2017 left off with the same specifications in mind. Specifically, the team is focussed on developing consistency with the application on all Android phones. The team also worked on how to make the app robust and simple for the user, i.e. user experience. The team brainstormed how to efficiently send data to a server that will be implemented and determine the necessary format. Spring 2018 is working on sending data using JavaScript Object Notation (JSON), which would allow data to be easily mirrored onto the website. As a result of thorough research, the WA Team discovered there are many different servers that can be used for the same purpose, thus resulting in further research to aid in deciding on which is more practical for the smooth transition of data.

5 Semester Documentation - Spring 2018

5.1 Team Members

Name	*Role*
Mohammad Jahanshahi	● EPICS Advisor ○ Advises EPICS Syllabus learning objectives ○ Technical and professional guidance
Margaret Phillips	● EPICS Advisor ○ Advises EPICS Syllabus learning objectives ○ Professional guidance
Seyedali Ghahari	● Teaching Assistant ○ Academic logistics and operations for EPICS section ○ Guidance in area of expertise - Civil Engineering ○ Assisted team in moving forward and finding resources
Erika Lin	● Project Manager - responsible for overall operation and effectiveness of team and provides planning, direction, and guidance ● Oversee HD, WA, and DA Teams laterally ● Design Documentation – ensuring whole team is appropriately documenting contributions and individual progress
Kartik Mittal	● App Development Design Lead ○ Oversees App design ○ Responsible for facilitating project through components of design process ○ Responsible for project planning, execution, risk assessment to deliver a quality team end-deliverable on time/budget ● Android Application ○ Finalizing the layout of the application ○ Implementation of server and storing the data ○ Google API implementation
Kalpan Jasani	● Data Analysis Design Lead ○ Oversees Data Analysis design ○ Responsible for facilitating project through components of design process ○ Responsible for project planning, execution, risk assessment to deliver a quality team end-deliverable on time/budget ● Data Analysis ○ Ensuring Kinect data collection (data management, error analysis) ○ Quantification of potholes

Romita Biswas	● Hardware Design Lead ○ Oversees Hardware design ○ Responsible for facilitating project through components of design process ○ Responsible for project planning, execution, risk assessment to deliver a quality team end-deliverable on time/budget ● Synchronization of Kinect Depth Data and GPS Data
Muhammad Shorieri	● Webmaster – Update and maintain project's website ● Learn Android Studio ● WA – GUI interface (front-end)
Rachel Lee	● WA – GUI interface (front-end) ● Documentation for the application
Khaing Thu (Helen) Zin	● WA – GUI interface (front-end) ● Ideas for layout ● Responsible for Transition Document
Kyaw San (Steven) Thway	● Project Archivist ● WA
Benjamin Hutchins	● Financial Officer – maintaining Spring 2018 team budget ● HD – designing POLES mounting frame
Brian Sutanto	● HD – Voltage source for Kinect unit. ● GPS and Kinect Integration
Ethan Tan	● DA – Simplification of code by breaking into simpler parts ○ Break down code into new files/functions ○ Create breakpoints in the code to allow for easier testing ○ Advise Hardware team for data input into code ○ Improve detection algorithm
Ayyub Jose	● DA – developing a mathematical model for pothole severity quantification
Dauzhan (Don) Yerkozhanov	● DA – Software implementation for data analysis ○ RANSAC, plane fitting, grayscale and Otsu, ○ Pothole severity quantification (bounding rectangle) ○ Reading data and outputting of severity analysis with measurements

5.2 Current Status and Location on Overall Project Timeline

The *Project Specification*, *Specification Development, and Conceptual Design phases have been completed. The Detailed Design phase is currently in progress.*

Spring 2018 Smart City began the semester working on understanding and developing previously developed hardware and software. Spring 2018 is focusing on consistent documentation of code as well as the design process. Previous decisions made in previous semesters were reconsidered and previous issues are to be resolved. Spring 2018 is continuing the *Detailed Design* phase by solving previous issues, presenting possible improvements, and implementing new features desired by the project partner.

At the end of Spring 2018 semester, we have a hardware system that takes depth data, but has not yet been tested on actual road conditions. There is a functioning severity analysis that has a few discrepancies. There is also a functioning website that needs to be tested with data from data analysis. The application is now fully functional on an Android device. However, the app needs to be tested on various Android devices and is currently only functional for pothole reporting.

5.3 Goals for the Semester

As a continuation of previous semesters, Spring 2018 is expecting to have a preliminary deliverable executable product. The HD team aims to be able to collect multiple continuous frames of data accurately timestamped. The DA team aims to be able to consistently accurately analyze each frame of data. The WA team aims to have a fully functional website displaying a map and pothole locations for the city as well as a fully functional application accessible to the public. Spring 2018 also discovered that previous code documentation was inconsistent and thus made it difficult to pick up where the team left off within a reasonable amount of time. As a result, a goal for the Spring 2018 semester will be developing a two-page manual for each team - HD, DA, and WA - such that following teams will be able to easily access and understand where the previous team stands at the end of the semester.

5.4 Semester Timeline

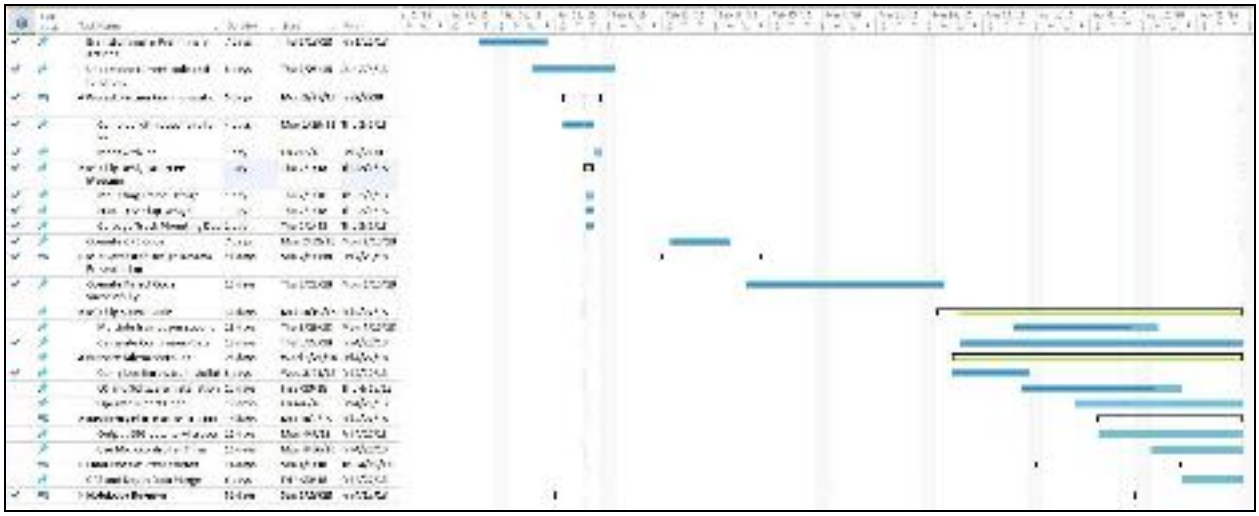
Smart City SharePoint>Semester Documentation>Spring 2018>Gantt Charts:

Week	*Design Process Milestones*	*Status/Notes*
1 — 01/11	Introduction to EPICs	• EPICS information/outcomes overview
2 — 01/18	Project Scope Review	• Smart City information/outcomes overview • Project roles assignment
3 — 01/25	Detailed Design	• Understand where previous semester left off • Brainstorming and timeline generation
4 — 02/01	Detailed Design	• Individual team cost-breakdown analysis • Team Website Update • Meeting with project community partner
5 — 02/08	Detailed Design	• Gantt Chart Update to Advisors and TA • Budget Analysis to EPICS Office

6 — 02/15	Detailed Design	• Mock Mid-Semester Design Review
7 — 02/22	Mid-Semester Design Review	• Design Document preparation and slides for Mid-Semester Design Review/Evaluation • Individual team testing/elaboration of available technology
8 — 03/01	Detailed Design	• Re-evaluate project based on Mid-semester review feedback • Revisit *Project Identification*, *Specification Development*, and *Conceptual Design*
9 — 03/08	Detailed Design	• Further design implementation
10 — 03/15	SPRING BREAK	
11 — 03/22	Detailed Design	• Further design implementation
12 — 03/29	Detailed Design	• Further design implementation
13 — 04/05	Detailed Design	• Final Design Review Presentation Slides • Design Document preparation for Final-Semester Design Review/Evaluation
14 — 04/12	Detailed Design	• Delivery checklist, if delivering • Design Document preparation for Final-Semester Design Review/Evaluation • 2-page Transition Documents for each team • Mock Final Design Review
15 — 04/19	Final Design Review	• Slides and Presentation preparation for Final-Semester Design Review/Evaluation • 2-page Transition Documents for each team
16 — 04/26	Final Write-ups	• Finalize 2-page Transition Document • Update Design Document with feedback
17 - 05/03	FINALS WEEK	**Design Process to be done** (see **Detailed Design**): • *Functional Prototype* (needs finishing) • *Functional Evaluation* • *Revised Prototyping* • *Field Testing* • *Drawing Conclusions & Recommendations*

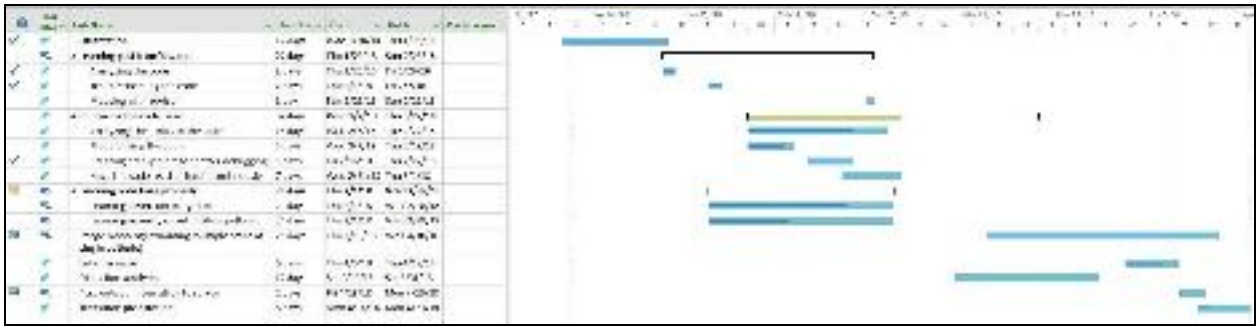
5.4.1 Hardware Semester Timeline

Task Mode	Task Name	Duration	Start	Finish
✓	Brainstorming & Preliminary Actions	7 days	Thu 1/18/18	Fri 1/26/18
✓	Understand current code and functions	8 days	Thu 1/25/18	Sat 2/4/18
✓	**Project Partner Communication**	**5 days**	**Mon 1/29/18**	**Fri 2/2/18**
✓	Come up with questions for PP	4 days	Mon 1/29/18	Thu 2/1/18
✓	Meet with PP	1 day	Fri 2/2/18	Fri 2/2/18
✓	**Modify Design after PP Meeting**	**1 day**	**Thu 2/1/18**	**Thu 2/1/18**
✓	Mounting Frame Design	1 day	Thu 2/1/18	Thu 2/1/18
✓	Frame Overlap Design	1 day	Thu 2/1/18	Thu 2/1/18
✓	Garbage Truck Mounting Desi	1 day	Thu 2/1/18	Thu 2/1/18
✓	Operate GPS Code	7 days	Mon 2/12/18	Mon 2/19/18
✓	**Mid Semester Design Review Presentation**	**11 days**	**Sun 2/11/18**	**Fri 2/23/18**
✓	Operate Kinect Code Successfully	18 days	Thu 2/22/18	Mon 3/19/18
	Modify Kinect Code	**30 days**	**Mon 3/26/18**	**Fri 4/27/18**
	Multiple frames per second	13 days	Thu 3/22/18	Mon 4/16/18
✓	Generate Continuous Data	27 days	Thu 3/22/18	Fri 4/27/18
	Prepare Microcontroller	**28 days**	**Wed 3/21/18**	**Fri 4/27/18**
✓	Complete Hardware Installat	8 days	Wed 3/21/18	Fri 3/30/18
	OS and Software Installation	15 days	Fri 3/30/18	Thu 4/19/18
	Operate Kinect Code	16 days	Fri 4/6/18	Fri 4/27/18
	Raspberry PI to NUC Microcontr	**15 days**	**Mon 4/9/18**	**Fri 4/27/18**
	Output Web data to Microcontr	15 days	Mon 4/9/18	Fri 4/27/18
	Use Microcontroller Time	10 days	Mon 4/16/18	Fri 4/27/18
	Final Review Presentation	**14 days**	**Sun 4/1/18**	**Thu 4/19/18**
	GPS and Depth Data Merge	6 days	Fri 4/20/18	Fri 4/27/18
✓	**Notebook Reviews**	**56 days**	**Sun 1/28/18**	**Fri 4/13/18**



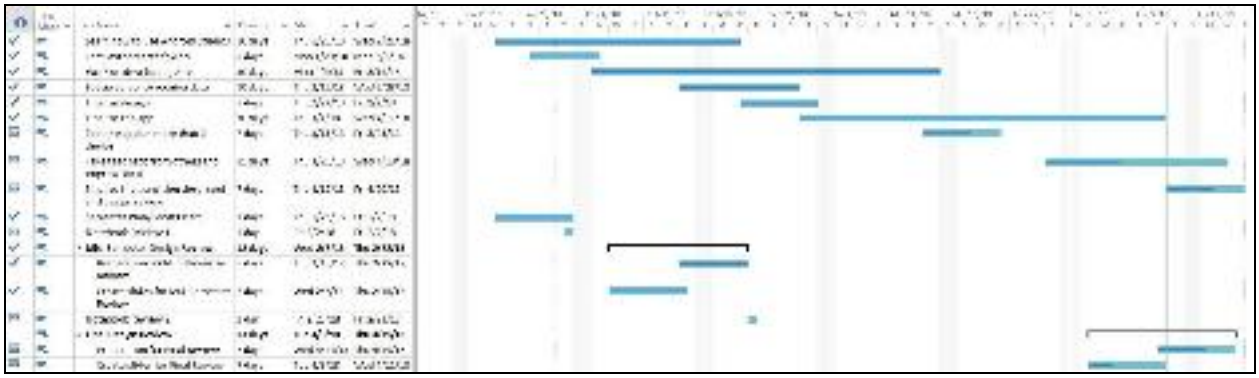
5.4.2 Data Analysis Semester Timeline

Task Mode	Task Name	Duration	Start	Finish
	Orientation	12 days	Wed 1/10/18	Thu 1/25/18
	Learning past team's work	**22 days**	**Thu 1/25/18**	**Sun 2/25/18**
	Analyzing the code	2 days	Thu 1/25/18	Fri 1/26/18
	Troubleshooting the code	2 days	Thu 2/1/18	Fri 2/2/18
	Meeting with advisor	1 day	Sun 2/25/18	Sun 2/25/18
	Improving the code base	**47 days**	**Wed 2/7/18**	**Thu 3/22/18**
	Analyzing the errors in the code	15 days	Wed 2/7/18	Tue 2/27/18
	Modularizing the code	5 days	Wed 2/7/18	Tue 2/13/18
	Creating breakpoints for better debugging	5 days	Fri 2/16/18	Thu 2/22/18
	Have the code read different continously	7 days	Wed 2/21/18	Thu 3/1/18
	Learning code base properly	**70 days**	**Thu 2/1/18**	**Wed 5/16/18**
	Learning detection in Python	20 days	Thu 2/1/18	Wed 2/28/18
	Learning severity quantification python	20 days	Thu 2/1/18	Wed 2/28/18
	Image mapping (trimming multiple scans of single pothole)	25 days	Thu 3/15/18	Wed 4/18/18
	Set up a server	6 days	Thu 4/5/18	Thu 4/12/18
	Distortion Analysis	17 days	Sat 3/10/18	Sat 3/31/18
	Pass output information to server	2 days	Fri 4/13/18	Mon 4/16/18
	Transition preparation	6 days	Mon 4/16/18	Mon 4/23/18



5.4.3 Website and Application Development Semester Timeline

Task Mode	Task Name	Duration	Start	Finish
	Learn how to use Android Studios	20 days	Thu 1/25/18	Wed 2/21/18
	Test last semester's App	6 days	Mon 1/29/18	Mon 2/5/18
	Work on developing App	30 days	Mon 2/5/18	Fri 3/16/18
	Set up server to receive data	10 days	Thu 2/15/18	Wed 2/28/18
	Finalize design	7 days	Thu 2/22/18	Fri 3/2/18
	Finalize the app	30 days	Thu 3/1/18	Wed 4/11/18
	Test the app on more than 2 device	7 days	Thu 3/15/18	Fri 3/23/18
	Take feedback from others and improve on it	15 days	Thu 3/29/18	Wed 4/18/18
	Finalize the transition document and design review	7 days	Thu 4/12/18	Fri 4/20/18
	Semester Plan/Gantt Chart	7 days	Thu 1/25/18	Fri 2/2/18
	Notebook Review 1	1 day	Fri 2/2/18	Fri 2/2/18
	Mid-Semester Design Review	**12 days**	**Wed 2/7/18**	**Thu 2/22/18**
	Preparation for Mid-Semester Review	6 days	Thu 2/15/18	Thu 2/22/18
	Create slides for Mid-Semester Review	7 days	Wed 2/7/18	Thu 2/15/18
	Notebook Review 2	1 day	Fri 2/23/18	Fri 2/23/18
	Final Design Review	**13 days**	**Tue 4/3/18**	**Thu 4/19/18**
	Preparation for Final Review	7 days	Wed 4/11/18	Thu 4/19/18
	Create slides for Final Review	7 days	Tue 4/3/18	Wed 4/11/18



5.5 Semester Budget

5.5.1 Proposed Semester Budget

TEAM BUDGET FORM

Team Name: SmartCity

Did you apply for a grant? (Circle)	Yes	No
Select if this form is being filed for the academic year or the semester. (Circle)	Year (17-18)	Semester

Project 1: Hardware Team

	Items required for project	Estimated Cost
1.1	Microprocessor	150
1.2	Cables	30
1.3	Glue	50
1.4	Frame	40
1.5		
1.6		
1.7		
1.8		
1.9		
1.10		
	TOTAL	$270.00

5.5.2 End of Semester Spending

TEAM BUDGET FORM

Team Name: SmartCity

Did you apply for a grant? (Circle)	Yes	No
Select if this form is being filed for the academic year or the semester. (Circle)	Year (17-18)	Semester

Project 1: Hardware Team

	Items required for project	Estimated Cost
1.1	Microcontroller	191.93
1.2	SSD	296
1.3	16 GB RAM	159.95
1.4		
1.5		
1.6		
1.7		
1.8		
1.9		
1.10		
	TOTAL	$960.88

Project 2: App Team

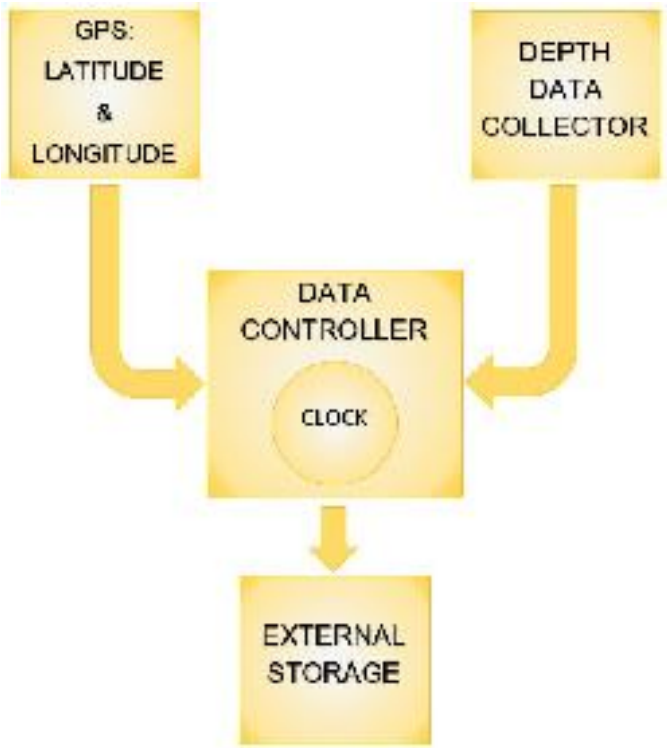
	Items required for project	Estimated Cost
2.1	Publishing App	50
2.2		
2.3		
2.4		
2.5		
2.6		
2.7		
2.8		
2.9		
2.10		
	TOTAL	$50.00

	Total Expenses
EPICS Beginning Allocation	-$200.00
Project 1: Hardware Team	$950.88
Project 2: App Team	$50.00
Project 3: Data Analysis Team	$0.00
Project 4: N/A	$0.00
Project 5: N/A	$0.00
Total Expenses	$1,000.88
Grants Requested	$0.00
Total Requested From EPICS Expenses Over All Projects	$800.88

6. Current Design

6.1 Hardware

For the Spring 2018 semester, the goal was to make the hardware system more cost effective and reliable by focusing on implementation on a specific type of city vehicle before eventually expanding. A decision was made to design the system with the Kinect sensor, and compatibility requirements allowed us to choose a new Intel microcontroller . Our goal is to successfully incorporate all the hardware components such as the GPS unit, Kinect, and operating system as a whole while also syncing the data collected with the accurate timestamps.



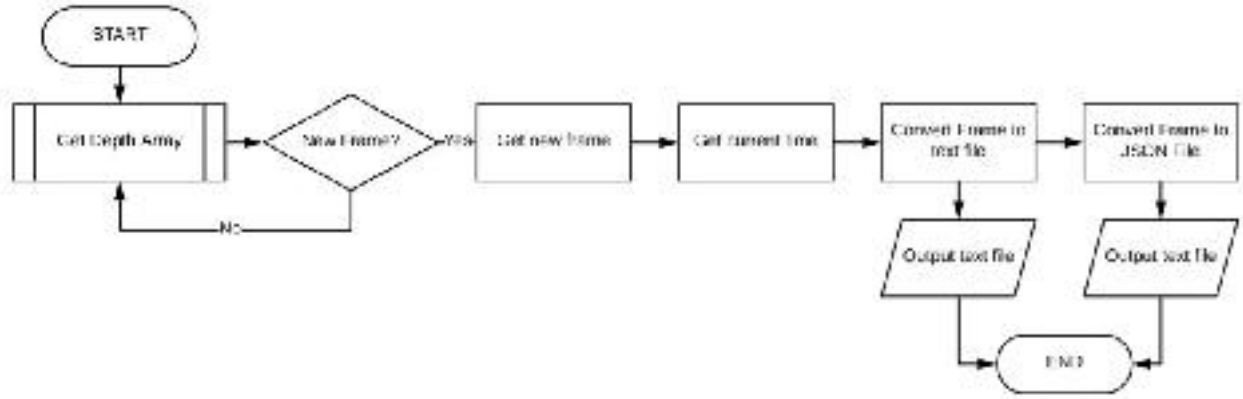
6.1.1 Kinect

Our current device that we are using for depth data collector is Microsoft's Xbox Kinect. This Kinect is commercially produced for the gaming industry, specifically for Microsoft's Xbox gaming system. The wide angle of the flight sensor within the Kinect is used to measure the distances by measuring the time taken by a light pulse. Even in minimum light environment, Kinect can still track objects at night through its IR sensor. Since UV rays significantly affects the Kinect's depth data output, we decided to run the Kinect on city vehicles at night at the optimal distance that Kinect can easily track, i.e. up to 4.6 feet. The depth data output will be collected at 1080 pixel resolution and 30 or 60 frame per second according to the chosen configuration.

The benefits and drawbacks of using Kinect include:

Benefits	Drawbacks
$81.00	UV ray sensitivity
IR sensor. At night.	Resolution up to mm
Time of flight sensor: measures distances from sensor	Small cracks can't be detected
Automatically generates a matrix	Distorted outer frame
60 frames/sec	

The basic functions performed in a single flowchart are shown below:



We have decided to use the Kinect's built-in Red Green Blue Depth (RGBD) sensor to detect depth data in the road. To test the Kinect sensor, we built fake potholes out of cardboard then held the sensor above the "potholes" to see if the sensor was able to detect the depth change. Using an application called Software Development Kit (SDK), the sensor displayed a clear distinction between depths. Microsoft has an application programing interface (API), but they provide advanced functions (facial recognition, heat detection, etc.) that our project scope does not require. We are utilizing SDK to tap into simple Kinect functions such as RGB image capture and depth data collection.

Figure: Artificial pothole; Early prototyping test of Kinect sensor.

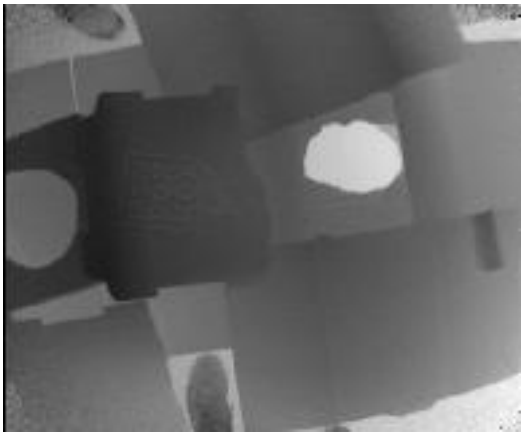


Figure: Screenshot of both potholes captured in one RGBD frame.

In the figure above, we can see the difference in depth from the RGBD sensor, which will be useful to the data analysis team when they try to program a function to determine depth.

Relative Weight	a	b	c	d	e	f	g	
Kinect Option	Field of View	Frame Per Second	Compatibility	Latency	USB	Min Depth Detectable	Score	
Kinect 1	57°	30	90	90	70	50	69.5238095	
Kinect 2	81°	60	70	60	90	55	70.1190476	
Sensitive	Field of View	Frame Per Second	Compatibility	Latency	USB	Min Depth Detectable		
Kinect 1	57.5° x 43.5°	30 fps		90 ms	USB 2.0	40 cm		
Kinect 2	70° x 60°	60 fps		60 ms	USB 3.0	50 cm		

Figure: Decision matrix for deciding to use Kinect Version 1 versus Kinect Version 2.

To decide on which version of the Xbox Kinect would be optimal for the purpose of this project, a decision matrix, as shown above, was used. In this project, we will using the RGB camera and Depth functions built-in to the Kinect sensor. The RGB camera on the Kinect functions like most digital cameras, i.e. grayscale color images, and the depth sensor is based on Infrared Emission[13]. Since our final product will be implemented outside, the sunlight will easily interfere the precision of depth sensor, and thus it is critical we put our product into a relatively concealed environment, like a box or chassis. This semester, we are attempting to use the vehicle as our box or chassis, as the main purpose is to prevent direct sunlight.

The goal of the hardware team is to detect and save pothole data such that the DA team is able to employ their pothole detection/quantification algorithm (see **6.2 Data Analysis**). We decided to save the RGB Image for future reference, e.g. calibration, error, data reference/benchmarking, etc. Overall, we decided we need to save following information:

- Original Depth data from Kinect Sensor (.txt)

**Note*: Kinect will be taking 1.2 MB of memory per frame. Each text file generated for each frame is about 1.2MB . Consider this requirement for future data collection implementation into the end-deliverable.

[13] https://molspect.chemistry.ohio-state.edu/institute/bernath4-IRemissionreview.pdf

The following is the suggested hardware requirement listed by Microsoft to run Kinect:
The following operating systems and architectures are supported:

- Windows 8 (x64)
- Windows 8.1 (x64)
- Windows 8 Embedded Standard (x64)
- Windows 8.1 Embedded Standard (x64)

Your computer must have the following minimum capabilities:

- 64 bit (x64) processor
- 4 GB Memory (or more)
- I7 3.1 GHz (or higher)
- Built-in USB 3.0 host controller (Intel or Renesas chipset).

**USB 3.0 functionality must support Gen-2.

- DX11 capable graphics adapter (see list of known good adapters below)
  - Intel HD 4400 integrated display adapter
  - ATI Radeon HD 5400 series
  - ATI Radeon HD 6570
  - ATI Radeon HD 7800 (256-bit GDDR5 2GB/1000Mhz)
  - NVidia Quadro 600
  - NVidia GeForce GT 640
  - NVidia GeForce GTX 660
  - NVidia Quadro K1000M
- A Kinect v2 sensor, which includes a power hub and USB cabling.

**When using Kinect Fusion, Kinect can process data either on a DirectX 11 compatible GPU with C++ AMP, or on the CPU, by setting the reconstruction processor type during reconstruction volume creation. The CPU processor is best suited to offline processing as only modern DirectX 11 GPUs will enable real-time and interactive frame rates during reconstruction.

Software Requirements:
The following developer environments are supported:

- All Visual Studio 2012, including Visual Studio 2012 Express (Microsoft Visual Studio 2012 Express)
- Visual Studio 2013 Ultimate, Premium, Professional, and Express for Windows Desktop

**Note*: Kinect's camera is 60 frames/second, meaning the Kinect can transfer up to 84 MB of data in one second, so each minute 5.04 GB of data can be generated. We suggest having the Kinect take frames that "overlap" instead of 30 frames/second (optimizing the frame capture to reduce required hard-disk space). Considering the amount of data collection per "session" (daily runs by vehicles the sensor system will be attached to), we decided to use make a revised hardware requirement:

- 64 bit Processor
- Physical dual-core 3.1 GHz (2 logical cores per physical) or faster processor
- USB 3.0
- 8GB RAM
- SSD Hard drive (500GB+) (faster data opening and writing)

For computer operating system specification, the official SDK only supports windows 8.1+. However, we could always use existing open source libraries and virtual machine implementation and run Kinect on any UNIX system, like MAC OS or Linux.

The following flowchart depicts how Kinect interacts with the computer:

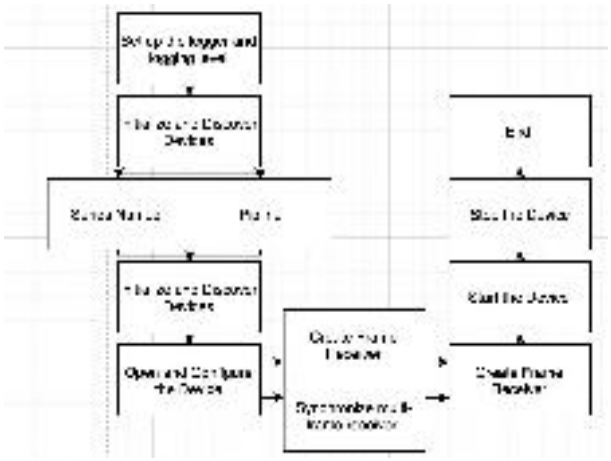


Figure: Flowchart of Kinect-computer communication

We have experimented with two programming languages: Python with open source libraries, and C++ with Microsoft SDK, which is more comparatively complex. SDK is designed for complex functions for human-computer interactions, i.e. skeleton, motion, or eye tracking. The complexity of SDK requires programmers to invest more time learning compared to Python. Spring 2017 had finished operating code in Python, and thus, Spring 2018 decided to continue utilizing Python while achieving synchronization in C++. Based on Kinect-computer communication, we are using following libraries and software:

- Python 2.7.13
- Opencv 3.2.0[14]
- Libfreenect2[15]
- Pylibfreenect2[16]
- XCode[17] and command line tool
- Terminal (pre-loaded on Mac OS)
- Kinect SDK Drivers

Initial Prototype and initial testing was performed on the following computer specifications:

- MacBook Pro (Retina, 13-inch, early 2015)
- Processor 2.9 GHz Intel Core i5
- Memory 8 GB 1867 MHz DDR3
- Graphics Intel Iris Graphics 6100 1536MB
- macOS Sierra Version 10.12.4
- Libraries and software above

The code can run on any computer with specifications:

- Libraries and software above

[14] https://github.com/opencv/opencv
[15] https://github.com/OpenKinect/libfreenect2
[16] https://github.com/r9y9/pylibfreenect2
[17] https://developer.apple.com/xcode/features/

Using the following provided code, we are able to build a Kinect-computer connection. The building process includes the following:

1. Open Packet Pipeline from CPU or GPU
2. Create Frame for future interaction
3. Create and set logger
4. Detect if Kinect device is correctly connected with computer
5. Get device serial number
6. Create Listener to receive data
7. If previous 6 steps work correctly, open Kinect device and officially build the connection

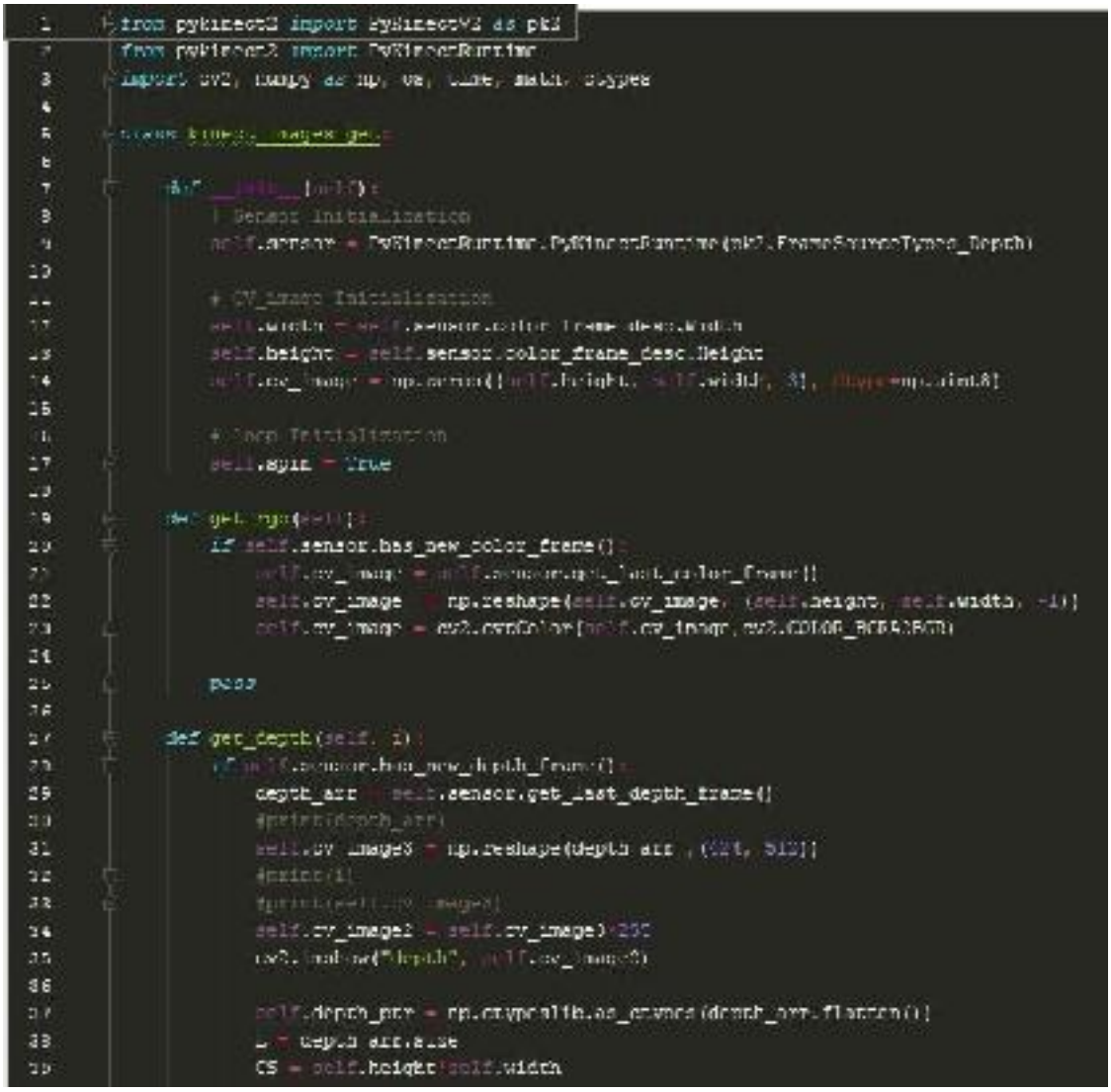


Figure: Source-code to establish connection/communication between Kinect and computer.

The next code will process and save the depth data, depth image and RGB image frame by frame. The following is detailed function of following code:

1. Create and redirect the directory where data will be saved
2. Stream depth video captured from Kinect
3. Stream RGB video captured from Kinect
4. Save depth data into txt file with following naming convention: number.txt
5. Repeating steps 1-4 permanently until Keyboard interruption (at this moment, you must manually terminate the program from running)
6. After Keyboard Interruption, stop the while loop, close the connection between Kinect, close the program and return to operation system

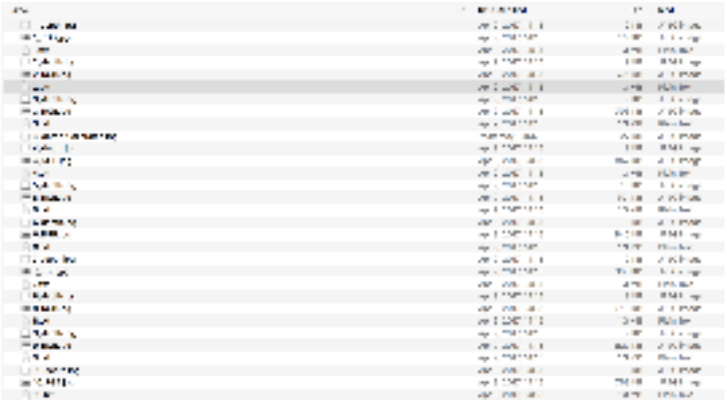


Figure: test depth data saved in 512*424 array as .txt, 1.3 MB. The depth Image (9 KB) and RGB image(731 KB) are saved in jpg file. The total size of one frame data is 2.04 MB.

In order to test how precise the Kinect is, we set up an error analysis test. The test was held on marble ground in the EPICS labs. Note: the error analysis test may be done on any surface but the calibration itself may be dependent on the surface material. Since the test was held indoors, sunshine and light is not considered for our error calibration test. We set up the Kinect at 6in, 9in, 12in, 15in, 18in, 21in, 24in, 27in, 30in heights. At each height, we took three frames and calculated the stability and precision of the data gathered by the Kinect. From our experiments, Kinect is effectively unable to get data from 6in, 9in, 12in, 15in and 18in. If a depth value is considered “invalid,” i.e. unreadable by the Kinect, then the Kinect depth sensor will return a zero value. From 21+ in., we are able to get valuable data.

In order to get a rough idea of Kinect’s precision, we start by calculating average depth. The Kinect is able to get 424 pixels by 512 pixels, i.e 217088 total depth data. Due to some changes in the code, we are now able to retrieve a 1080 x 1920 HD resolution and provides a wider frame, thus more accurate values. However, the tradeoff is a higher image size. The output text files are 14mb in size, which is an issue moving forward. Due to reflection and other independent random variables, Kinect occasionally yields invalid measurements. After exclusion of invalid data, we calculate the average depth and the number of measured distance (depth from Kinect to surface).

The following form shows the result of average measured distance and percent of error from each test:

Actual Distance(in)	Test Number	Measured Distance(in)	Percent of Error	Average Percent of Error
21	#1	20.85	0.71	2.77
	#2	21.93	4.44	
	#3	21.67	3.17	
24	#1	24.10	0.48	1.01
	#2	24.23	0.97	
	#3	23.61	1.64	
27	#1	26.99	0.04	0.53
	#2	26.80	0.76	
	#3	27.21	0.79	
30	#1	29.44	1.85	2.96
	#2	28.50	5.01	
	#3	30.59	1.98	

Figure: Sample error fitting test.

For this project, our project scope is detection and quantification of potholes (assumed normally deeper than 0.40 inches). Comparison to an assumption of 0.4 inches, average percent of error for Kinect depth collection is smaller than 1.67%. In the test table above we can see that 24in and 27in meet the requirement (precision of Kinect is relatively stable within this range). We calculated the normal distribution the data we gathered in 24 in and 27 in:

- For 24in test, 54.15% of the data fall in the range of 23.60in to 24.40in (Percent of error equals 1.67% range), and 97.70% of the data fall in the range of 23.20in to 24.80in (Percent of error equals 3.33% range).
- For 27in test, 55.96% of the data fall in the range of 26.60in to 27.40in (Percent of error equals 1.67% range), and 98.57% of the data fall in the range of 26.20in to 27.80in (Percent of error equals 3.33% range).

**Note*: *We* suggest that, end-deliverable should be installed within this range (24in to 27in) for accurate data collection.

A test Kinect housed in a sealed box to reduce light influences took measurements calibrated at 25in:

Figure: RGB image for pothole detection test

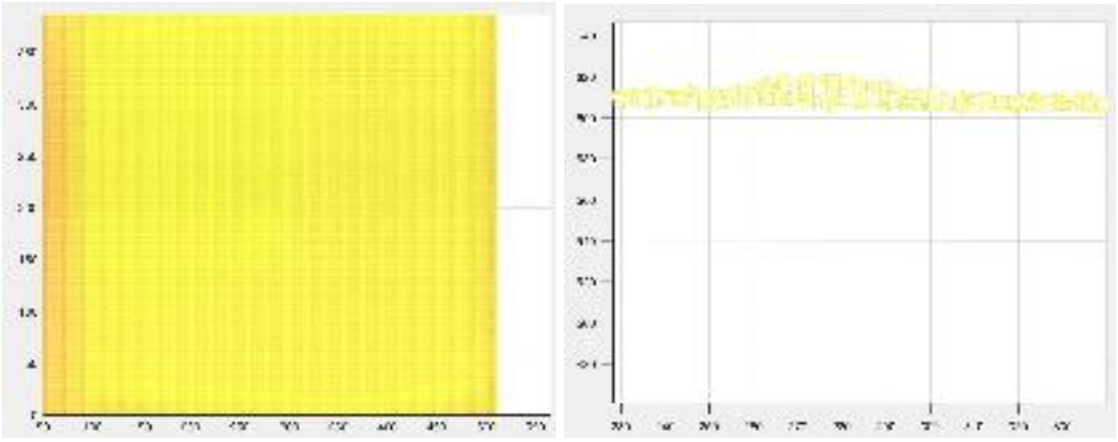


Figure: MATLAB adjusted RGB image based on depth data; first: top view of depths; second: front view of depths (vertical axis: depth, horizontal: width)

It was decided that since the amount of data generation was a big data issue that rgb images would be destroyed upon generation of the depth data text files to minimize any unnecessary files and reduce processing speeds of the python code in order for the desired fps to be achievable. Spring 2018 iteration decided to remove the RGB function and collect only depth data text files. Data Analysis team did not require RGB data either, so it was best to focus on collecting depth data text files continuously. The kin1lite.py code was manipulated to do so. Currently the data is collected at a set rate of 20 fps but it can be manipulated based on the vehicle speed and amount of necessary overlap.

As mentioned above, the Kinect sensor data is influenced and changed by the sun's UV rays. According to the design in Fall 2017 documentation, they were using only one single Kinect attached to the bumper of the truck shown in figure below.

Figure: Field of view with one Kinect implemented.

One Kinect limits the field of view to a minimal and limited area, thus unable to gather data on the entire road area. After calculations of the Kinect field of view relative to the Kinect height and road area, we concluded that implementing three Kinects would be the most effective for gathering thorough data. The figures below demonstrate the area in which multiple Kinects will cover:

Figure: Field of view using three Kinects.

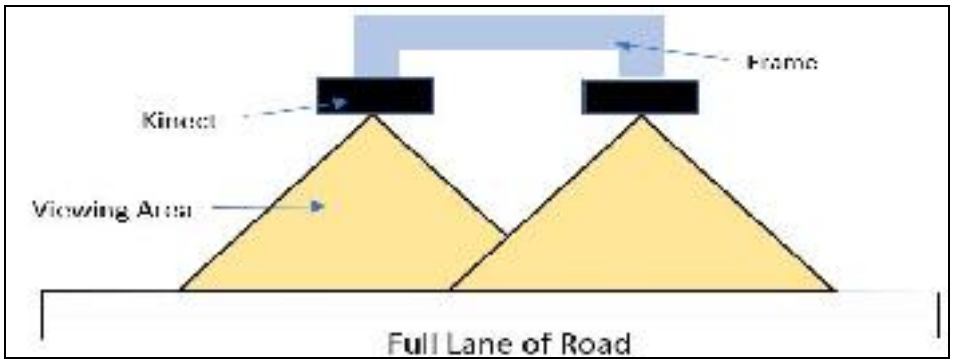


Figure: Field of view using two Kinects.

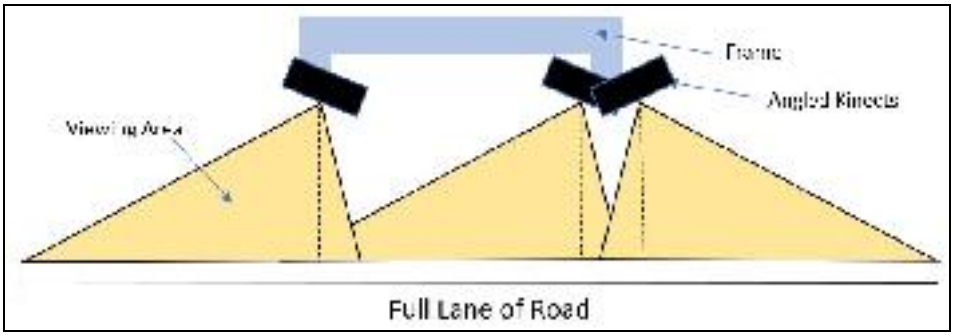


Figure: Field of view using three Kinects.

The Kinect housing system was built to house all the hardware and block the sunlight's interference with the Kinect sensor. The box is to be constructed of a lightweight polymer with a flexible curtain of rubber wrapped around the base. However, Spring 2018 has decided that this prototype would be an inefficient way to collect data with such a large model, which may also result in slow gathering of data. Therefore, we are no longer pursuing the Kinect housing system and will be implementing the Kinect beneath the city vehicle, as if the vehicle itself acts as a housing system as shown in the figure above.

See 6.1.4 Addressing the Issue of Direct Sunlight for more information.

6.1.2 Raspberry Pi

In Fall 2017, the Hardware team planned to use the GPS and Kinect to integrate both the systems on a shared microcomputer Raspberry Pi 3. The latest version, Raspberry Pi 3, is more efficient in terms of speed and memory. However, with the Raspberry Pi 3, we will only be able to operate the first version of Kinect instead of Kinect v2 that we were using in the semester of Spring 2018. Since the new Kinect v2 offered higher resolution output, it required a higher specifications in the Operating System. Therefore, we decided to use the microcontroller from Intel to run the Kinect.

Figure: The Raspberry Pi assembled with Screen

Specifications of Raspberry Pi 3:

- SoC: Broadcom BCM2837
- CPU: 4× ARM Cortex-A53, 1.2GHz
- GPU: Broadcom VideoCore IV
- RAM: 1GB LPDDR2 (900 MHz)
- Networking: 10/100 Ethernet, 2.4GHz 802.11n wireless
- Bluetooth: Bluetooth 4.1 Classic, Bluetooth Low Energy
- Storage: microSD
- GPIO: 40-pin header, populated
- Ports: HDMI, 3.5mm analogue audio-video jack, 4× USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)

6.1.3 GPS

For the Spring 2018 HD Team, the goal includes collecting GPS data (capture time and latitude/longitude coordinates) for potholes detected and quantified by **6.1.4 Data Analysis**. GPS data could tell our stakeholders the location of potholes and be given recommendations on which roads to fix.

The Fall 2017 GPS sub-team's first goal was to choose a sensor that is small, portable, but more importantly, weatherproof. GPS sensor will most likely be placed on the outside of a vehicle (despite plans for housing), so a non-weatherproof GPS unit would most likely break. We chose Adafruit Ultimate Breakout[18]. Adafruit Breakout can be used through a USB port while Internet connection is not needed (continue receiving data when the vehicle is running). Data collection can be synchronized based on **6.1.1 Kinect** (to optimize and minimize data collection). We will be executing the GPS on Raspberry Pi 3, which has a Linux system, but we can also run the GPS on Mac OS-X and Ubuntu.

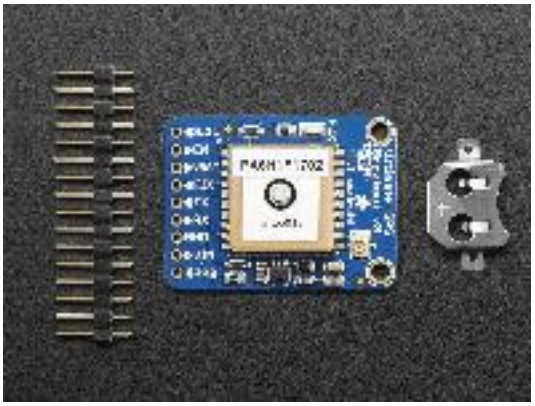
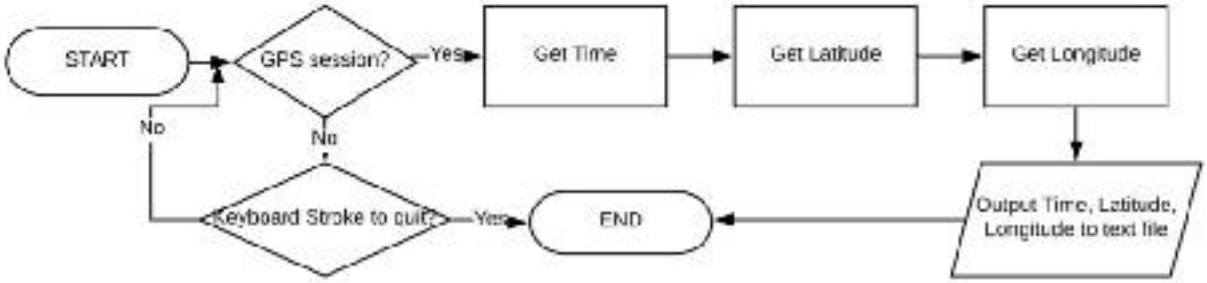


Figure: Adafruit Ultimate Breakout GPS



[18] https://cdn-shop.adafruit.com/1200x900/746-13.jpg

The specifications of GPS system and Raspberry Pi are as follows:

Specifications:

1. Hardware Specifications[19]

- Size: 1 x 1.4 x 0.3 inches and Weight: 0.3 ounces
- -165 dBm sensitivity, 10 Hz updates, 66 channels
- 5V friendly design and only 20mA current draw
- Breadboard friendly + two mounting holes
- RTC battery-compatible
- Built-in datalogging
- PPS output on fix
- Internal patch antenna + u.FL connector for external active antenna
- Fix status LED

2. Software (Raspberry Pi integrated specifications)

- Linux environment on Raspberry Pi 3
- Code is on Python 2.7.1, which gives output in a text file
- Does not require internet to work
- Outputs 1 data point (position and time) per second
- Can take 20 seconds to 15 minutes to find a fix (at least 3 satellites), but when facing a clear sky, it finds a fix in generally 7-10 seconds

3. Instructions to access the code for GPS

- The code is currently both on Raspberry Pi and Sharepoint too
- For Raspberry Pi, the code can be found on the location: Pi>Home>gps>gpscode1.py
- Some sample tests (GPSTest1, GPSTest3) are also available in the same folder
- These documents are available on Sharepoint under Project Documentation too

Data will be automatically collected in a ".txt" file after starting the code (data store locally to a specified hard-disk location). We found a source code to run the GPS, which was modified to limit the output fields to just latitude, longitude and timestamp.

[19] https://www.adafruit.com/product/746?gclid=CjwKCAiA9f7QBRBpEiwApLGUio5zkULI2DUHdHUMpp68_czMz6bQ8KWUcdLUZdYN6SmXhizpIizf1RoC0O4QAvD_BwE

The following image shows the modified code:

```
import gps

# Listen on port 2947 (gpsd) of localhost
session = gps.gps("localhost", "2947")
session.stream(gps.WATCH_ENABLE | gps.WATCH_NEWSTYLE)

fileName = 'GPSTest.txt'
fileOpened = open(fileName,'a')

while True:
    try:
        report = session.next()
                # Wait for a 'TPV' report and display the current time To see all report data, uncomment the line below
                # print report
        if report['class'] == 'TPV':
            if hasattr(report, 'time'):
                print report.time
                fileOpened.write(str(report.time) + '        ')

        if report['class'] == 'TPV':
            if hasattr(report, 'lat'):
                print report.lat
                latitude = report.lat
                fileOpened.write(str(report.lat) + '        ')

        if report['class'] == 'TPV':
            if hasattr(report, 'lon'):
                print report.lon
                 longitude = report.lon
                fileOpened.write(str(report.lon) + '\n')

    except KeyError:
                pass
    except KeyboardInterrupt:
                quit()
    except StopIteration:
                session = None
                print "GPSD has terminated"
```

Figure: code for gps data collection

Fall 2017 was unable to run the GPS outside, hence the above code output only a stream of timestamps, which can be seen in the figure below. When tested outside, the GPS will output latitude, longitude and timestamp data, an example of which (at one timestamp) is also shown below.

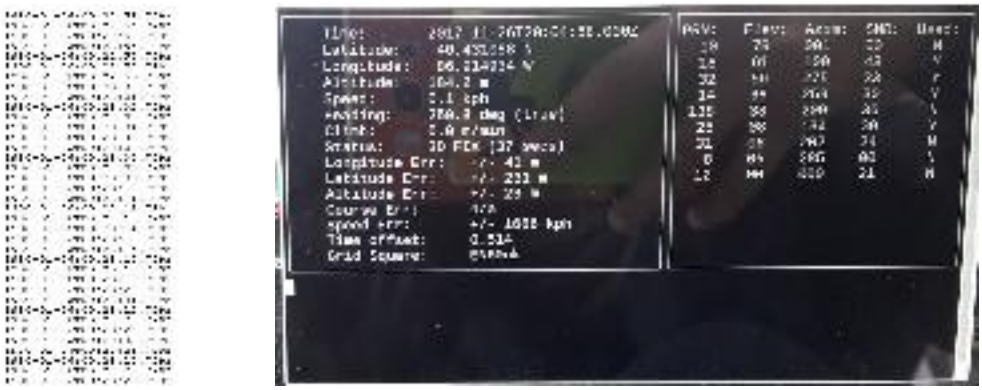


Figure: Time output in text file and all fields data for one timestamp

Spring 2018 picked up where Fall 2017 left off. We discovered that two files of code existed for the Kinect 1 and Kinect 2 as the previous semester attempted to execute both to decide which would be more efficient. We have decided on further developing the Kinect 2. Thus far, we have primarily focused on documentation of code since deciphering previous semesters' code was inefficient.

In order to run the GPS it had to be connected to the microcontroller through the Raspberry Pi since the GPS was only compatible with the Raspberry Pi, so the NUC microcontroller and the Pi are connected through a server so that the NUC can extract GPS data onto the microcontroller and have the same clock time to ensure synchronization of the data. A different GPS that is compatible with windows will have to be chosen so that extraction is easier. An issue with the current method of extraction is that a shared IP address is required meaning that an internet connection is required at all times which is not viable with the current prototype. However a server system may be the best to pursue since the data generated could be sent to a server in real time.

6.1.4 Kinect Mounting System

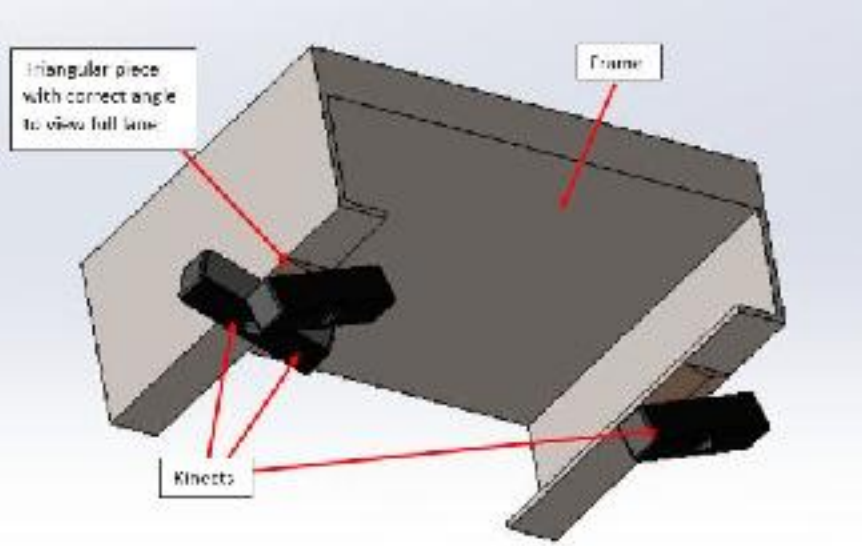


Figure: current design for the three Kinect 2 system covering a full lane of road.

The previous semesters came up with a design to attach a housing compartment for the mounting system to attach to the back of a car. This idea was chosen to prevent sunlight from interfering with the device. Through conversations with our project partner it was determined that we could run the device at night where this would not be a problem. The design that the hardware team has chosen as a potential design for future semesters is to attach a system of three kinects in order to cover an entire lane of road. This was chosen since for this semester we are focused on attaching the device to garbage trucks and not small vehicles. The two potential locations of the frame to attach the device is the middle frame of the truck and the front frame of the truck. The most ideal place to attach the device would be in the front near the engine since we would be able to power our device from the battery of the car. This current design doesn't account for a compartment that is needed to store the GPS, Raspberry Pi, and storage. An obvious way to attach this device to the frame would be through screws; however, there is a concern that drilling into the frame could damage the structural integrity of the frame. this semester we are just focused on getting one kinect to function properly.

Kinect Angle and FPS Calculations

Several calculations were done to determine the ideal angle of the potential 3 kinect system as well as the frame rate that the code must run in order to thoroughly capture the road conditions. The team used geometry to calculate the angle of the kinect, with the height off the ground being 2.5 feet and the distance from the outside tire to the frame being 2.625 feet. We then used the Kinect's viewing angle of

70 degrees. The results that we found were that the outside kinects would have to be angled at 20.41 degrees from the horizontal to cover a full lane of road (assuming that the average lane width is 10 feet) and found the angle to be 11.40 degrees to viewing just under the truck

The next set of calculations done were to determine the frame per second that the code must run to collect sufficient data based off of the height, viewing angle, and speed. For typical garbage truck operations, it is recommended that the code is ran at 20 frames per second.

Height off ground (ft)	2.5
Total viewing angle (degrees)	60
Variable Speeds (mph)	FPS
10	5.081
20	10.161
30	15.242
40	20.323
50	25.403
60	30.484

Figure: This figure shows the minimum frames per second that the code must run so that the frames overlap at various speeds.

Engineering Drawings

From our project partner site visit we measured the height off the ground of the frame to be two feet six inches. From this information we found out that if we attached the kinect vertically on the frame it would not cover the width of the truck or of a lane. From this information we determined that we would need to angle the kinect. We also determined that we would need three kinects to cover an entire lane as shown in the CAD drawing. Shown above are the dimensions of the triangular piece designed that would go on the frame. Regarding the material of this component, two materials we considered were PLA or ABS plastic and 6061 Aluminum. We determined that using a three-dimensional (3D) printer to create this part would be preferable as it would be more difficult to machine the angle on a mill with precision.

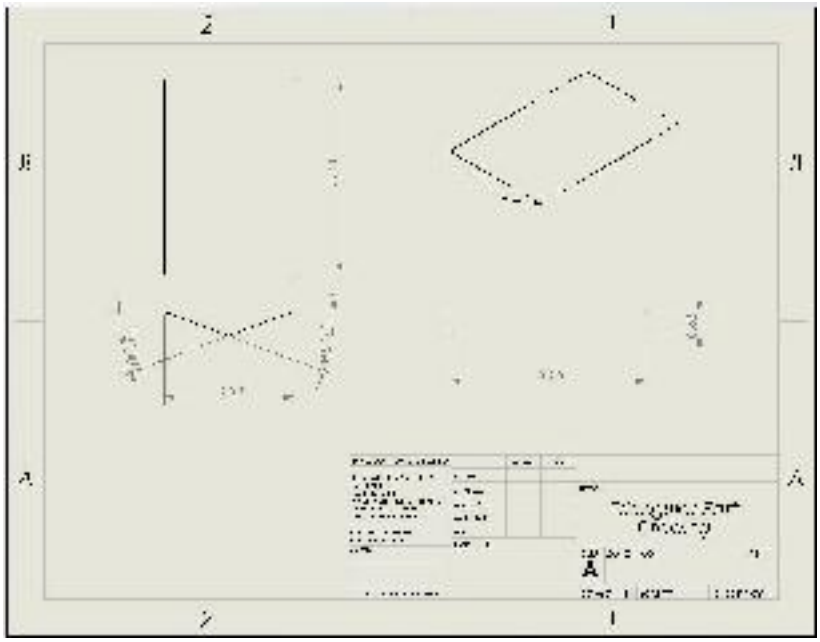


Figure: Drawing of the triangular piece of the mounting system

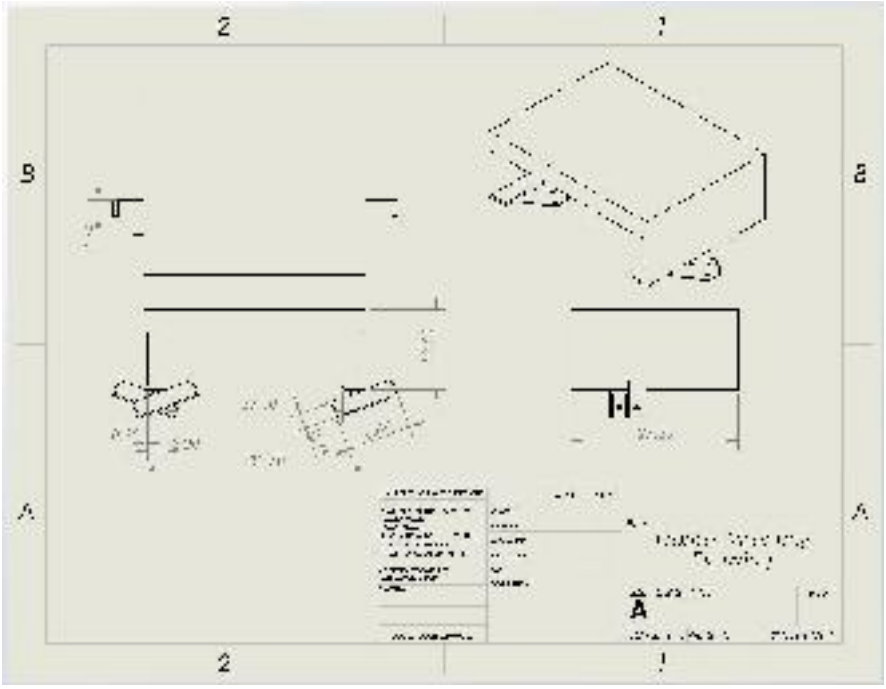


Figure: Dimensioned view of the assembly with the Kinects attached

Utilizing Natural Shadow from Vehicle

After much research and deliberation, Spring 2018 decided to forgo this idea and instead use the shadow of the vehicle to block out direct sunlight as the overall goal is to implement the sensor beneath the city vehicle. For Spring 2018, we decided to pursue the design with a specific city vehicle, i.e. garbage trucks.

6.1.5 City Vehicles: Garbage Trucks

Spring 2018 decided to implement a design such that the sensor would be placed beneath the truck using the vehicle shadow in place of the previously implemented box. Another option is that we can mount the Kinect onto the front bumper, since they have holes and spaces that would allow us to install the mounting system easily. After the project partner visit, we learned that there are various types of garbage trucks and constraints to how the device is permitted mounted on the vehicle itself. Below are images from the project partner visit, displaying the different types of garbage trucks available:

Figure: Garbage Truck 1a

Figure: Garbage Truck 2

Figure: Garbage Truck 1b

Figure: Garbage Truck 3

These city garbage trucks all have different features and dimensions, and we concluded that there is no one solution that would be applicable to all three. Thus, we decided to construct our design off of one type of garbage truck - Garbage Truck 3. Below are images of this garbage truck:

Figure: Bottom Right

Figure: Front Bumper

Figure: Right Side

City Garbage Truck Specifications

- The garbage truck that was chosen to base the design off of is a 1996 model and has a height of roughly 11 feet with a width of 8 feet.
- The frame in the middle and in the front is 2 feet 6 inches off of the ground.
- The frame is 2 feet 7 ½ inches from the outside of the tire on both sides. The frame has two pieces that come down on each side that are 2 feet 9 ½ inches apart. The frame component is 3.5 inches wide.

6.1.6 Intel NUC7i7BNH Microcontroller

Spring 2018 has decided to purchase a new microcontroller designed by Intel to improve speed of data processing. Intel NUC7i7BNH is also proven to be compatible with Kinect 2 after research from different forum pages and comparing the specifications of the microprocessor and the system requirements for Kinect 2. The feature for Intel NUC7i7BNH is listed below:

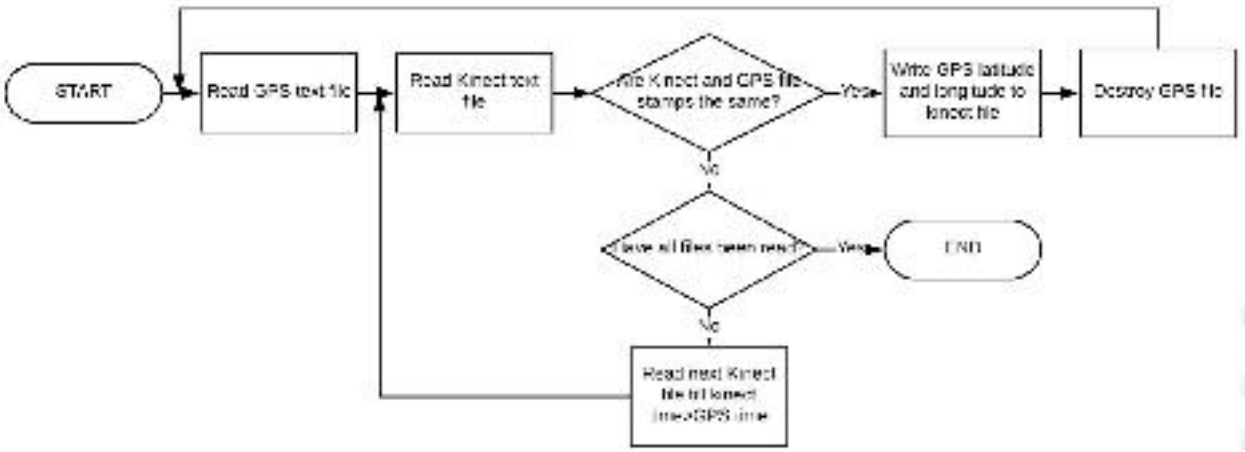
- Intel Iris Plus graphics
- Intel Optane memory
- Thunderbolt 3
- HDMI ports
- Support for 2.5” drives
- Intel Dual Band Wireless (802.11ac) and Bluetooth 4.2
- 4 USB 3.0 ports
- Intel Gigabit LAN
- Micro SD card slot

However, to run the Intel NUC and install the operating system into the microcontroller, we still needed a DDR4 SO-DIMM RAM and a 2.5” SATA SDD. As a result, we purchased Samsung 960 PRO Series - 512GB PCIe NVMe - M.2 Internal SSD (MZ-V6P512BW) as our memory storage drive. For RAM, we purchased Samsung 16GB 2133 MHz (M471A2K43BB0-CPB) for our RAM card or physical memory module.

Figure: Intel NUC7i7BNH (left), Samsung 960 PRO Series - Internal Solid State Drive with PCIe connection (right)

Figure: DDR4 RAM Samsung 16GB 2133 MHz

Once we have set up the microcontroller, it will be operating with the Kinect as well as the GPS to synchronize all the collect data with the correct time stamps. The critical task of the microcontroller is to match up the depth data which will be collecting at 60 fps and GPS at 10 fps with the appropriate time stamps. The details of the synchronization process can be explain with the following flowchart:



6.1.7 Final Design Review Comments/Reflection

- Address/further research difficulties in synchronizing three Kinects
    - Angles Kinects? Overlapping?
- How do you account for the motion of the vehicle?
- Address why there would be a difference between the different city vehicles.
    - At next project partner visit, measure and gather information regarding exactly why they are different.
- Would the system require any action on the part of the vehicle operator?
- How to handle acceleration: maybe explore using accelerometer after system works.
- How to keep lens clean during operation
- Can UV filters be used to allow daytime use?
- How to connect to battery safely, even if temporary?
- Address big data issues.
- Address how the tradeoffs are right for the project specifications: mention project partner needs parallel to the design.
- Address why the housing was eliminated and reasoning behind that decision.
- Using edge processing to minimize big data?

6.1.8 End-of-Semester Summary

We have completed significant commenting and improvement of the code. The code outputs a continuous stream of depth data. Currently it is at a fixed fps that can be changed manually, but the calculations are given for the next semester to implement a variable fps depth collection. We also found a microcontroller that was compatible with the Kinect 2 and have prepared it to be completely operable with the kinect code. The GPS we currently have is only compatible with the Raspberry Pi, but a different GPS whose data can be extracted can easily be used. Currently we access the GPS data files from the microcontroller through a server which requires a shared IP Address. The major issue we had was transitioning from the previous semester as no transition document was given for the kinect code, so we prepared a very detailed transition document for the next semester for all of the components of our system. Romita's Notebook shows what are the next steps and ideas for what to do the next semester to improve the code and manipulate it to operate at a variable fps.

6.2 Data Analysis

The primary role of Data Analysis is to take the input offered by the hardware team, and then process it, and finally pass it forward.

6.2.1 Requirements:

The Data Analysis team is responsible for:

1. Accepting hardware team's input
2. Developing and maintaining code to perform analysis
3. Detecting whether potholes exist
4. Determine severity of pothole
    a. Identify one pothole in multiple images (Image tracking)
    b. Analyze distortion of a pothole
5. Pass the output to a server
6. Prepare a transition for the next year's team

6.2.2 Overall Data Analysis Process

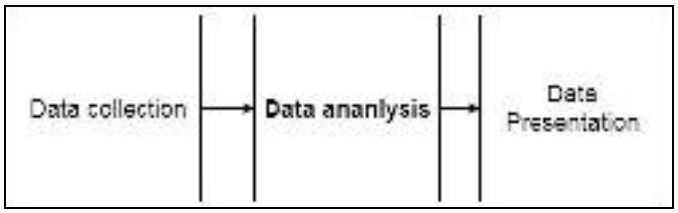


Figure: Context of the team

The data analysis team takes input as text files from the hardware team, processes them, and provides a smaller data of the frames with the potholes for further display.

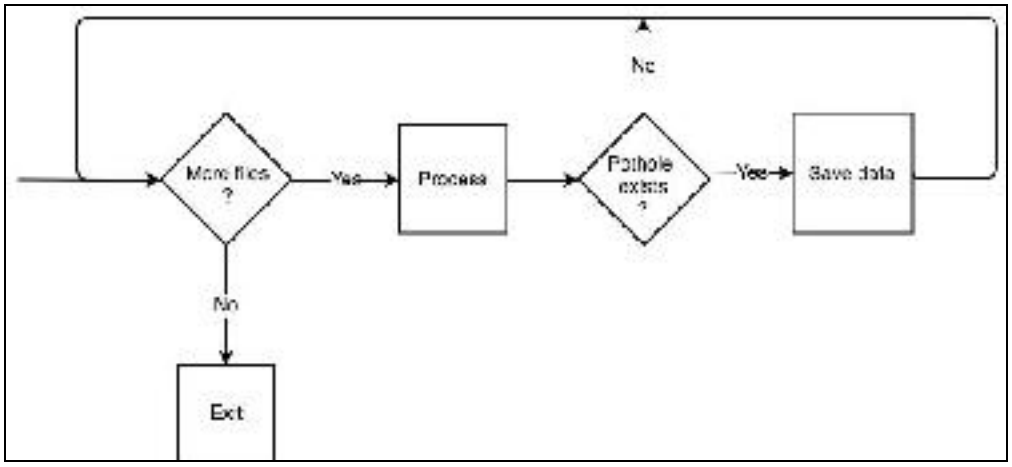


Figure: Overall process of the Data Analysis team.

Our flow chart demonstrates how our code continuously reads the data that Kinect captures (given by Hardware team), until the last file is read.

6.2.3 Current Method Adopted by the City

In the United States, various road surface condition inspection guidelines have been developed by various regulatory agencies. The City of West Lafayette currently uses the Pavement Surface Evaluation and Rating (PASER) Manual developed by the Transportation Information Center at the University of Wisconsin-Madison. This approach rates road conditions from a 10 (excellent) to a 1 (failed) and takes into account four factors: surface defects, surface deformation, cracks, and potholes/patches. But, potholes are only relevant for 3 of the 10 levels. We discuss the approach for severity that we take, in later sections.

6.2.4 Our Intended Approach

The task of our team is to automate the entire process of detection and quantification of the severity of potholes.

Our advisor, Jahanshahi, had readily available resources to develop an algorithm automating road-pavement defect detection and quantification (Unsupervised Approach for Autonomous Pavement-Defect Detection and Quantification Using an Inexpensive Depth Sensor)[20]:

The steps to be taken for this algorithm are:

1. Fit a plane to raw depth data using random sample consensus (RANSAC)[21] algorithm.
2. Subtract values from fitted plane to get a matrix of relative depth
3. Use Otsu's method[22][23] to find a threshold value to define the pothole's location in the frame
4. In the depth matrix, utilize the information available using Otsu's binarization to further determine the severity.

From this, we adapted to create a procedure which is depicted in the figure below.

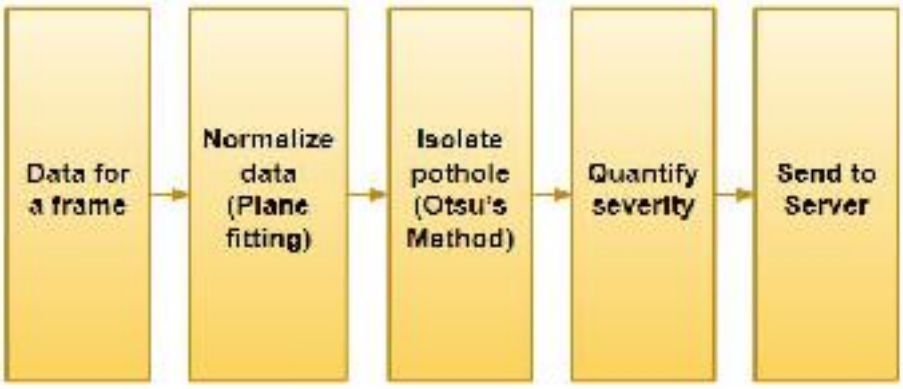


Figure: Figure explaining the process adopted for processing

[20] http://ascelibrary.org/doi/abs/10.1061/(ASCE)CP.1943-5487.0000245
[21] https://www.mathworks.com/discovery/ransac.html
[22] http://www-cs.engr.ccny.cuny.edu/~wolberg/cs470/doc/Otsu-KMeansHIS09.pdf
[23] https://www.mathworks.com/help/images/ref/graythresh.html

Note: Our entire implementation is done as a Python program. The location can be found below in the end of the section.

6.2.5 Plane Fitting

Why plane fitting?

With plane fitting, we define the road in our data. The road is a plane, and we then find the depth of other pixels, relative to this "road".

We have implemented the RANSAC (Random Sample Consensus) algorithm for plane fitting in Python. Our plane fitting code is modified from that developed from a third party person, whose code and explanation was available on Github[24].

Here are some of the images that explain the results from plane fitting.

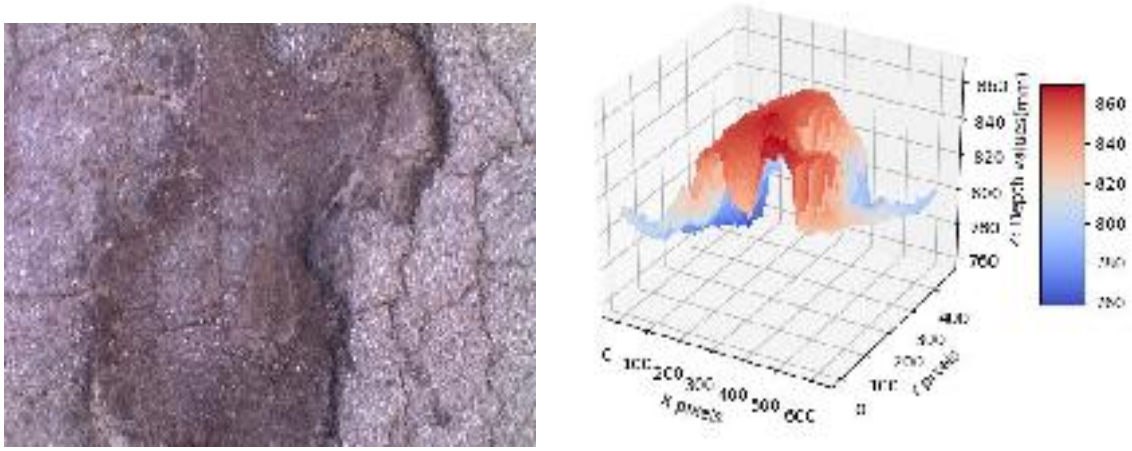


Figure: Pothole image 13_130 and its corresponding depth data from Kinect

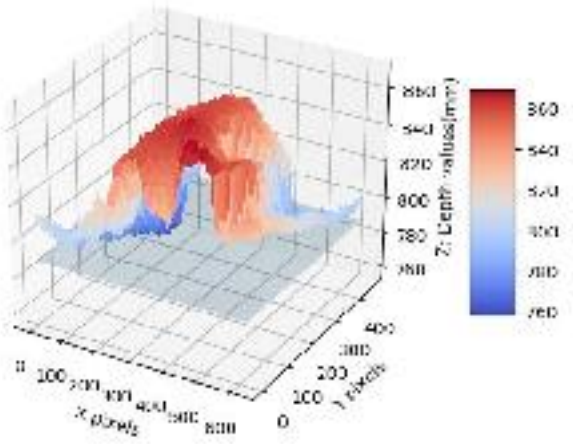


Figure: Plane fitted to original depth data

[24] https://github.com/minghuam/point-visualizer/blob/master/point_visualizer.py

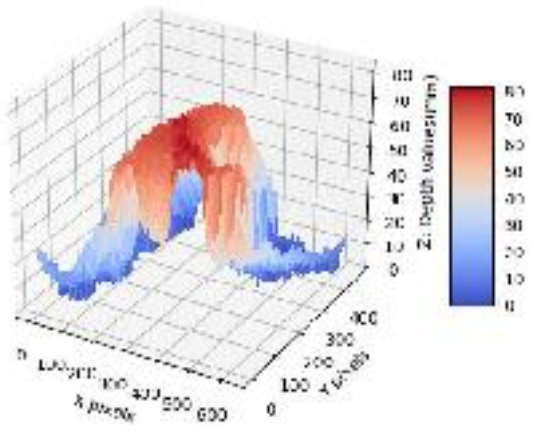


Figure: Normalized data

Note that in the last figure, the values are about 80 mm. The z value represents the depth at that x and y location. Due to calculation of relative depth, we now have a depth value that is relative to the level of the road.

The code where we implement plane fitting is in the file leastSqCoefficient.py We have commented this file to explain the different aspects of the code.

A reference for RANSAC is: http://www.cse.yorku.ca/~kosta/CompVis_Notes/ransac.pdf

Steps:

- Make a collection of all the points (pixels) that are a certain distance away from the pre known distance of the road. For our data, we kept this distance of 780 mm. Say we call this set S, for "set". We will only fit a plane using this S, and not the other points in the frame.
- Choose a certain number of points from this set S. This subset should be representative of the entire set S. To do this, we can randomly select points, or select points at in jumps, so as to select the whole frame in some sense.
- Choose the "threshold", the "tolerance" and the "number of iterations" for the RANSAC algorithm
- Perform RANSAC for plane fitting on this set of points. We will obtain a plane, given by its equation.
- Get relative depths from the plane.

Some design choices:

1. Selecting a sample of points from all the points in set S discussed above.
    a. Choose random set of points: Say you want to choose 1000 points. We can obtain 1000 random points.
    b. Choose points at certain jumps: By jumps we mean gaps in the frame. To see why this is better, imagine a case when random sampling does not provided a good result. For example, if we pick lot of points from one area, and that area is slightly depressed in the frame, our results could be affected. If we have points at regular intervals, we can obtain a better representation of from all the points available. That is, the data is evenly distributed.

We chose to go with the first approach as that is easier to implement and does not impact (hopefully) the representation of the road when the number of points to be picked are a large amount. But remember that the second approach isn't necessarily better.

Points to note:

- We also have to set all points that are a certain distance from the plane, to be 0. This helps remove noise from the data. This is as prescribed by the research paper we use for our project (Unsupervised Approach for Autonomous Pavement-Defect Detection and Quantification Using an Inexpensive Depth Sensor [25]).

6.2.6 Otsu's Binarization

Otsu's is a method to divide the data set into two parts. In application to our project, it uses the frequency distribution of the depth to find the threshold to differentiate the two parts of the image. These two parts are:

1. Road
2. Pothole

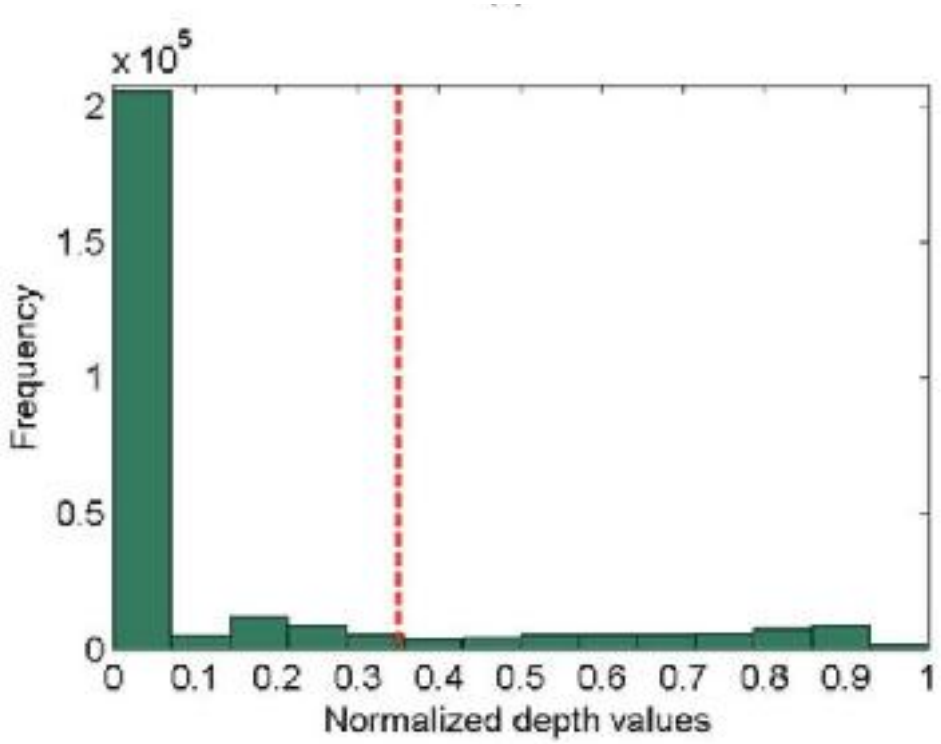
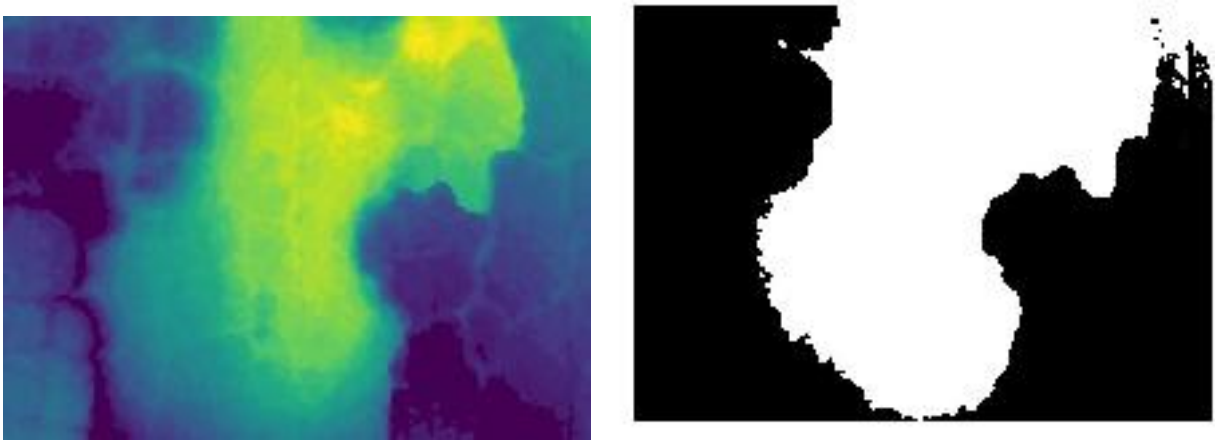


Figure: Demonstration of Otsu's method

Assume we have a frequency distribution as given above. The depth values are all from 0 to 1, as they are normalized, or they are all relative depth values to the maximum depth in the frame. As can be seen, there will be a high frequency for the category of depth value of 0 to 0.1. This makes sense because there will be a lot of points that are of the road. Otsu's method will result in one depth value that will differentiate the set of points.

[25] http://ascelibrary.org/doi/abs/10.1061/(ASCE)CP.1943-5487.0000245



Figures: Relative depth(left) and result after Otsu's binarization(right)

The left figure is showing a thematic representation of the depth values in the image, after plane fitting. The right image represents the "output" after we get a depth from binarization. As can be expected, we can clearly differentiate two distinctly different parts in the left image, and that results in Otsu's binarization in the right.

The white areas represent depressed region. After we have received this information, we have a clear pothole outline obtained from Otsu's binarization and we are able to move on to defect quantification.

6.2.7 Quantification of Severity

We designed several approaches to quantify the severity of the potholes. According the paper of Professor Jahanshahi that we follow for plane fitting and Otsu's, there is a scheme for quantification of the severity of the pothole. This was actually attempted during the first semester of this team. But, for the semester of spring 2018, in continuation from the Fall 2017 projet, we adopt the rectangle bounding scheme for quantification of the severity.

Note: There has been significant variation and choices in the past semesters on choosing a good model for quantification of severity.

In the end, we aim to give results based on standards provided by an established organization. For this, we researched on the following available.

Discussion of Standards:

1. PASER rating: This is adopted by the city for manual inspection currently. This is mentioned in the section "Current method adopted by the city".

We needed a way to classify potholes alone, rather than the general condition of a road. We found two alternative guidelines for pothole quantification (in terms of its width and depth) from:

2. the U.S. Department of Transportation Federal Highway Administration (FHWA)

Maximum Depth of Pothole [mm]	**Minimum plan dimension [mm]**
	150
< 25	Low
25 - 50	Med
> 50	High

Table: Pothole severity as defined by FHWA[26].

3. American Society for Testing and Materials International (ASTM):

Maximum Depth of Pothole [mm]	**Average Diameter of Pothole [mm]**		
	100 - 200	200 - 450	450 - 750
13 - 25	Low	Low	Med
25 - 50	Low	Med	High
> 50	Med	High	High

Table: Pothole severity as defined by ASTM International[27].

Based on ASTM standards, to be able to quantify how severe the pothole is, we require the average diameter of the pothole and also the maximum depth of the pothole. After compensating for any errors, the maximum depth of the pothole is obtained using the smallest value in the depth array.

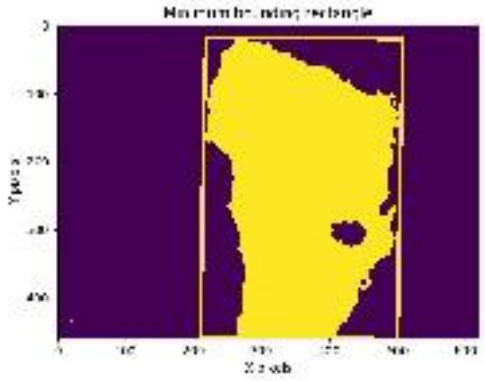
Determining Average Diameter:

One approach for calculating the average diameter is to calculate a minimum bounding rectangle for a pothole, and use the length and breadth to get the average diameter of the pothole.

Minimum Bounding Rectangle

In our python codebase, the findContours[28] method is called on the binary threshold of the pothole. This will give us a contour of the pothole, which is a curve joining all continuous points. Now, we would need to fit a minimum bounded rectangle, which considers the rotation of the rectangle as well. We can use function cv2.minAreaRect()[29] which will return a 2D box structure containing the center, width of pixels, height of pixels and angle of box rotation.

[26] https://www.fhwa.dot.gov/publications/research/infrastructure/pavements/ltpp/99168/99168.pdf
[27] http://www.oregon.gov/ODOT/TD/TP_RES/ResearchReports/AsphConcertePatch.pdf
[28] https://docs.opencv.org/3.3.1/d4/d73/tutorial_py_contours_begin.html
[29] https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html



Now, we will convert from pixel units to real world units. We already have the horizontal field (an angle), vertical field (an angle), resolution (number of pixels on each dimension) and the height of the Kinect. This is pre given to us when were working on sample data provided by our professor (see note). Obtaining the length of one dimension of the image, *l* can be done by using formula:

$$l = 2h \tan(\alpha/2)$$

h = height of kinect, α = vertical field or horizontal field.

Note: Even in our project we suppose that we will have a fixed height of the Kinect.

After obtaining the length of both sides of the image, we divide the length by the number of pixels in the image to obtain the size of the individual pixel. As we know the width and height of the pothole bounded by the rectangle (in units of pixels), we multiply it by the individual pixel lengths to obtain the actual length of the rectangle. To obtain the average diameter of the pothole, we add up both of the lengths and divide it by two.

The severity of the pothole is then evaluated using if-else statements based on the ASTM pothole severity table in the image below

Maximum Depth of Pothole [mm]	**Average Diameter of Pothole [mm]**		
	100 - 200	200 - 450	450 - 750
13 - 25	Low	Low	Med
25 - 50	Low	Med	High
> 50	Med	High	High

Table: Pothole severity as defined by ASTM International[30].

[30] http://www.oregon.gov/ODOT/TD/TP_RES/ResearchReports/AsphConcertePatch.pdf

We are also able to determine if the depth image contains a pothole or not by using the maximum depth and average diameter. If the average diameter is less than 100mm and the maximum depth is less than 13mm, we can consider it not a pothole by ASTM standards based on the ASTM severity table above.

Important: This is a crude way of telling whether a pothole exists. We can definitely come up with better designs for determining whether a frame contains a pothole or not. Some designs that we looked at:

1. Before performing binarization, we find the maximum depth, and if that is too little (< 13mm), we can say that the frame does not contain a pothole)
2. After doing Rectangle bounding: Use average diameter and maximum depth to say if there is a pothole.

Difference in approaches	Before binarization	After rectangle bounding
Processing time of program	We can save processing time	We perform this check in the end
Accuracy ● False positive: Detecting when there isn't ● True negative: Not detecting a pothole when there is a pothole	Lesser	Higher

Figure: Table for comparison of approaches on answering the question of whether a pothole exists.

6.2.8 Visualization and testing of implementation

We utilized a software called Tracker to measure pothole dimensions. This is for the sole purpose to test our Rectangle Bounding algorithm. The testing for the pothole below gave a matching result. The figures below are screenshots to show the working on Tracker software. Again, while Tracker is not important for the objectives of our team, it was a tool to accurately tell different lengths in the manual measurements we took.

Figures: Measurement of the pothole

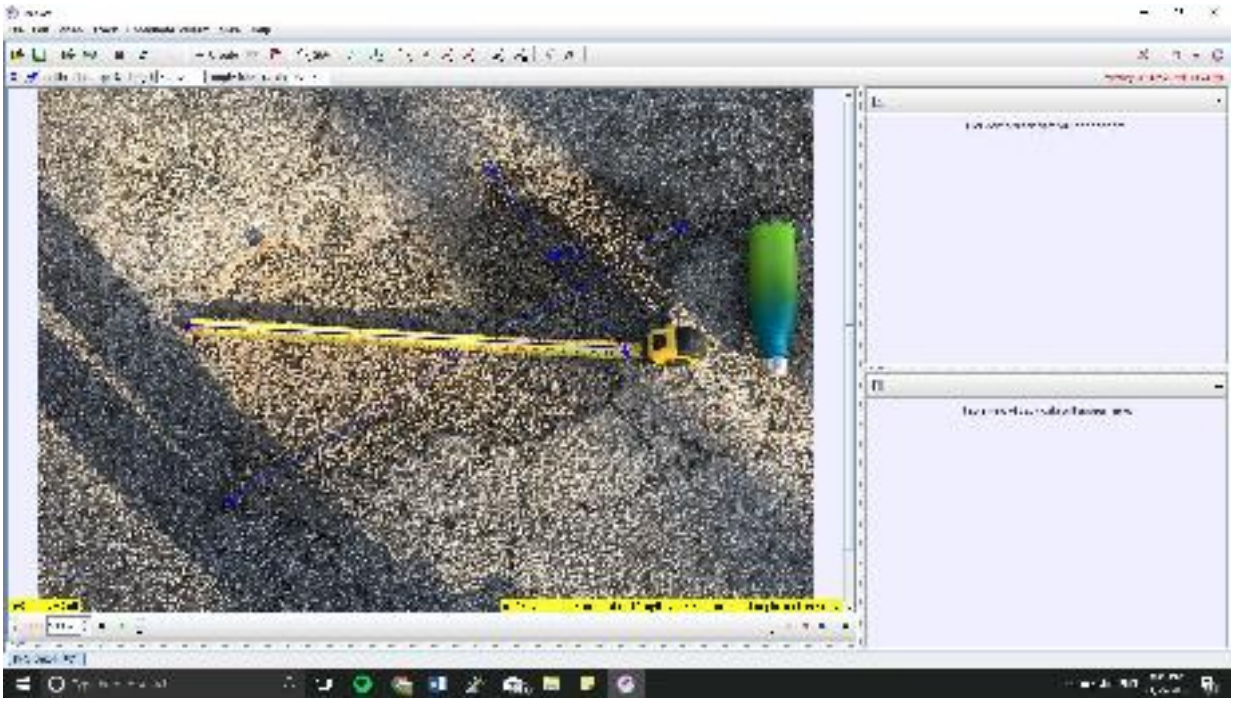


Figure: Screenshot taken from the software for reference

6.2.9 Location of execution of program and data transfer

An important issue is:

> "The speed of the data analysis program is slow, while data collection is done in real time."

Therefore, we cannot process each file in real time. We need to store the data from the hardware team. This made us decide on the following options on running our program.

	Running on a server	Running on Microcontroller	Running on facility
Maintenance (1-10)	7	9	5
Data transfer across network (1-10)	2	8	8
Storage (1-10)	4	4	4
Speed of processing (1-10)	9	3	7
Total	22	24	24
Rank	1	2	2

Table: Decision matrix behind the choice of place of program execution

6.2.10 Location where our code can be found and tested

Our current, most up-to-date python code integrating all of the functions to receive and analyze an image, can be located in https://github.com/Kalpan-Jasani/SmartCitySpring2018. Do look at the README.md file to further utilize the same

6.2.11 Data Tracking

An important aspect of this current semester was to develop and image tracking software that could identify the pothole from video data. It was important to get isolated data and get the perfect pothole possible. The current program is iterative. You input one image or a sequence of images and its it calculates the severity of those potholes frame by frame. If you have a sequence of pothole it has no identifier and treats each image of the pothole as a separate entity. Pothole data will obviously be sequenced between frames when u are capturing data at 20 frames a second in a moving car , the data will be segmented and you will be calculating the same pothole many times and different incomplete segments of the same pothole. This was a main side task this semester and it was important in moving the project forward.

6.2.12 Current Approach

There is not actually a current approach right now and this is something new we are currently testing. We hope to have a prototype at the end of the semester. Having a prototype will be a good foundation for the upcoming semesters.

6.2.13 Our Methodology

The goal is to create a video tracking algorithm that can identify, track and isolate a pothole in a video file or a sequence of images. The data we get from the hardware team i a sequence of both depth images and jpeg files. Tracking can occur using either data but at the moment we have the image processing techniques to detect a pothole using depth data, we would have to create a whole new algorithm to detect potholes using RGB data. Detecting pothole from RGB is less accurate than using depth, you can get the same results but there is more room for error. I initially planned to use depth data as the basis for my tracking but i decided to make the switch to RGB.

6.2.14 Contour Tracking

Why Contour? From Ayuub Jose's Senior Design Project

Contour tacking is a way to track and detect movements object movement in a cluttered environment. The Condensation algorithm seeks to solve the problem of estimating the conformation of an object described by a vector. This method of tracking uses the boundary contour of a moving and deforming object in a sequence of images. In the first instance, the contour of the object is obtained in the first frame, for us this is the pothole. Once, a rough contour of the desired structure is available on the first image of the sequence, the system automatically outlines the contours on the subsequent images at video rate. Contour based Object tracking is useful in many areas such as motion based recognition. Contour tracking should work here, our pothole is a road deformation, after we carry out plane fitting and Otsu's Method on the depth stream, the pothole is defined as a contour. With the algorithms' predictive nature, this should theoretically be the right approach. Contour tracking is very good for well defined shapes, and it works well with edge detection software, which is key in separating the pothole from the road.

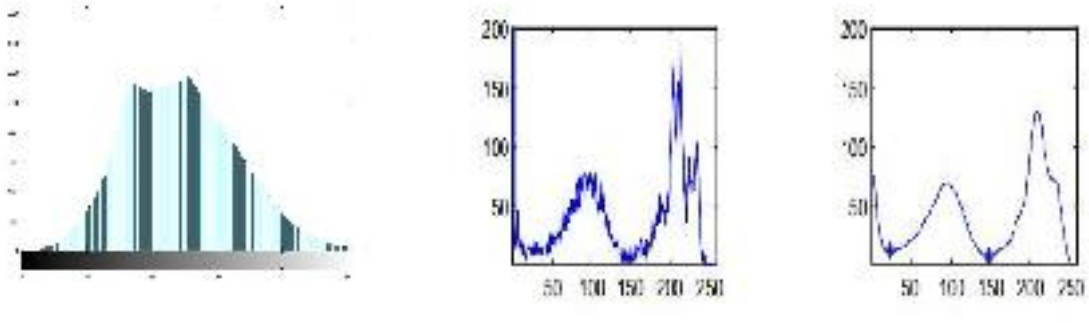
With thresholding we will extract the pixel of the pothole from its background.

$$T = T[x, y, p(x, y), f(x, y)]$$

Above is the general function for thresholding(T), where (x and y where the coordinate) pixel coordinates nad f a global function. They can all be used in tandem or each part separately depending on how apply the thresholding.

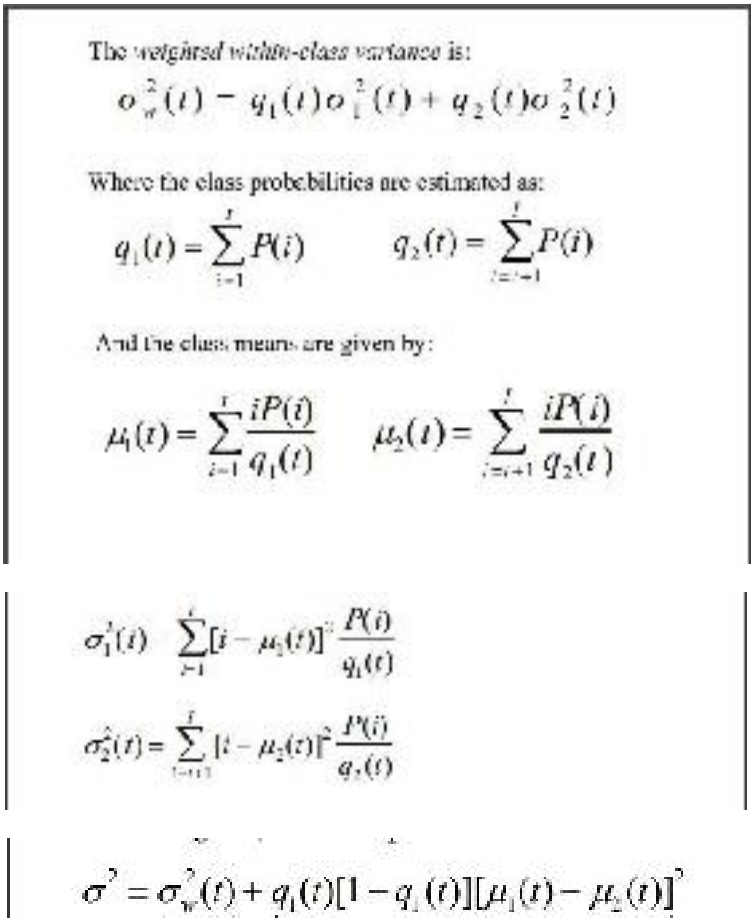
There are three main ways for thresholding: Global Thresholding, which uses valleys in the histograms of pixels, Fixed Thresholding, where a fixed point is used, and Optimal Thresholding, where you find the intersection of two different peaks.

It was important to look at the benefits of each thresholding method and pick the best one that could fit the project's needs. From the get go, Global Thresholding was not going to work as Global thresholding works well for bimodal histograms, which are histograms with 2 modes (2 peaks) and the threshold value is the lowest point in the valley between the peaks. My pothole histograms have only one peak so there is no valley to choose between peaks.



To the left is a histogram of out pothole for testing and on the right is a histogram that uses global thresholding.

In my program, I use a mix of fixed thresholding and optimal thresholding.. Histogram shapes are not always reliable for threshold selection when peaks are not clearly resolved. – A "flat" object with no discernable surface texture,and no colour variation will give rise to a relatively narrow histogram peak, which is a good description of a pothole. So we used otsu's method to isolate the pothole. Otsu's methods works differently here than it does in depth as explained in the sections adobe. When using depth data, you have a 3D representing information, so the data can be visualized with a mesh function. With Otsu's depth data, it is an easy differentiation since you already know what value is the ground and the potholes. With RGB its based on finding the threshold that minimizes the weighted within-class variance. which is the same as maximizing the between-class variance, the calculations operates directly on the gray level histogram [e.g. 256 numbers, P(i)]. • I've used it with considerable success in "murky" situations.



The weighted within-class variance is:

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t)$$

Where the class probabilities are estimated as:

$$q_1(t) = \sum_{i=1}^{t} P(i) \qquad q_2(t) = \sum_{i=t+1}^{I} P(i)$$

And the class means are given by:

$$\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)} \qquad \mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{q_2(t)}$$

$$\sigma_1^2(t) \quad \sum_{i=1}^{t} [i - \mu_1(t)]^2 \frac{P(i)}{q_1(t)}$$

$$\sigma_2^2(t) = \sum_{i=t+1}^{I} [i - \mu_2(t)]^2 \frac{P(i)}{q_2(t)}$$

$$\sigma^2 = \sigma_w^2(t) + q_1(t)[1 - q_1(t)][\mu_1(t) - \mu_2(t)]^2$$

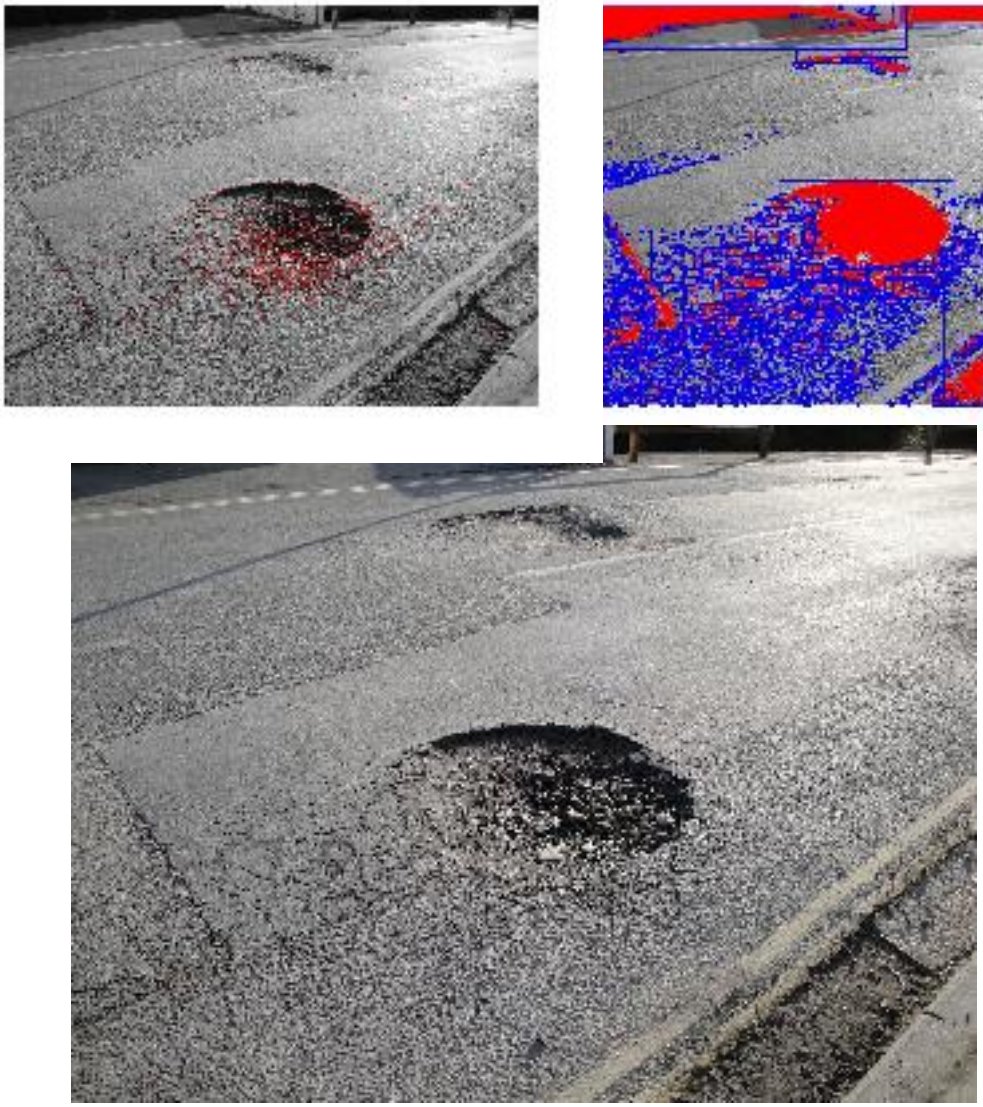
These are the calculations for getting the threshold value which is based on the total variance of the pixels. When using other programs this may have to be written up or modified but when using matlab you can calculate this value with ‘graythresh’ which is what I did in the program.

Object Tracking

For image tracking, I initially wanted to approach the program using foreground detection. With the input of the binarized image I can initialize that the background is anything false and only logic values of true are the pothole. Then using MATLAB video tool i can find the potholes in the video file then have a centroid to isolate and count the number of potholes. This method can be applied in future semester when depth video data can be collected

For image tracking I tried using the same function used for the single frame. I made a step function that tried to locate a pothole in each frame, then when it was able to fit the centroid within the field of view it would count how many centriods where in the data file and that would be the number of pothole. So it was important for me not to remove small centroids that factored as noise. i was able to make an alpha that counts the uses my pothole detection code and applies it ot video and coundts the centriods found in the image and identifies them as the pothole. I have run into problems that were not encountered in single image processing. It does find the pothole centroid bu there are no more unidentified centroids that just seem to come at random.

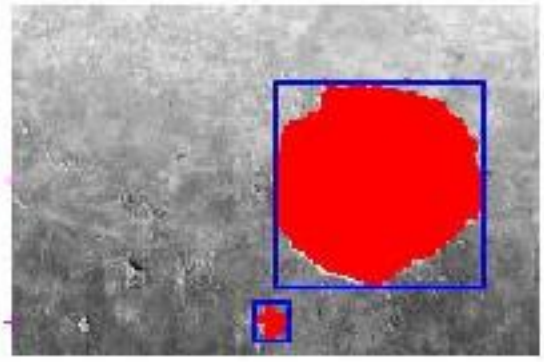
Results



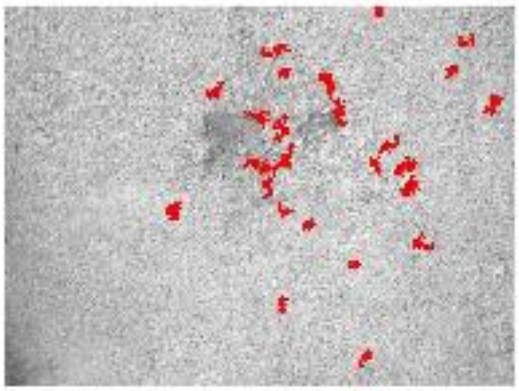
The pothole shown above is a good example of what I spoke about before. I have three images; at the bottom the original pothole image and two different phases in testing.The program works decently well here; in the first image to the left as it had identified the region where it thinks a pothole is but cannot really detect the edges, though it cannot detect the other pothole in the image. I made some changes and added a centroid to add proper definition to the pothole. In the image to the right it has been able to fill the pothole with red and i was able to draw the centroid though they data is more sensitive this time round, it can detect the pothole in the background but a lot of additional information was added, it's now very sensitive to cracks and the shadows in the far end of the picture. This pothole would rank as very severe on the astm and even with processing through depth those tiny cracks would also affect the information gotten.

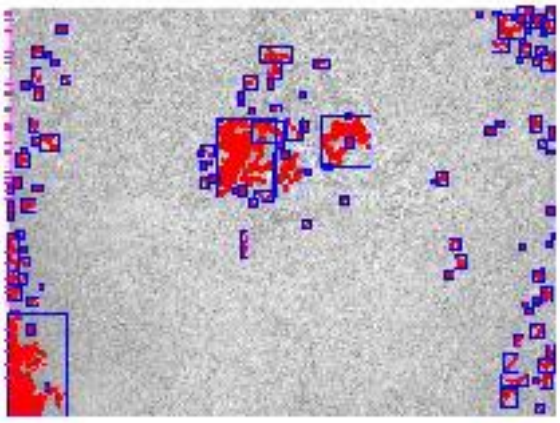
I should have added a function to remove centroids below a certain size to make the data more clean cut.. Ignoring the data marked as red at the top of image should be taken into account as it is the dark background of the picture.

I am making somes assumption that a reason for error is due to the angle at which the picture is taken and the problems with lighting. Our pictures/video is taken at a 90 degree angle so i do not expect these problems to persist though more testing is needed. More work is needed on thresholding as at this moment it was manual and I had to looks at the histogram to provide a range of values.

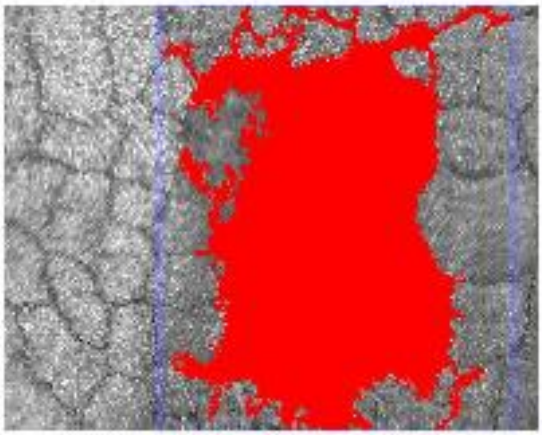


Here is a good example of the centroid working well in this test image, this image is controlled testing and seems to be a man made deformation, it is good we get perfect values here because the regions are well defined

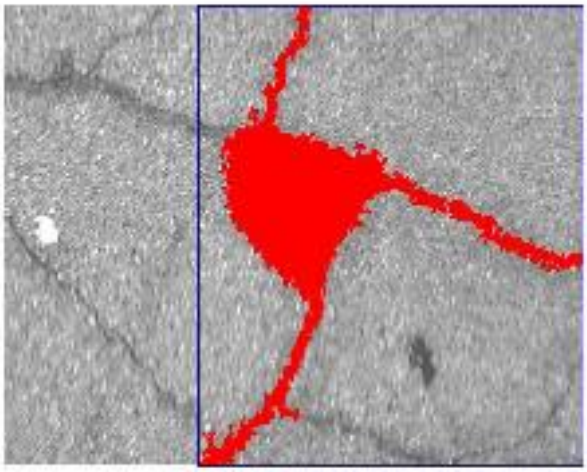




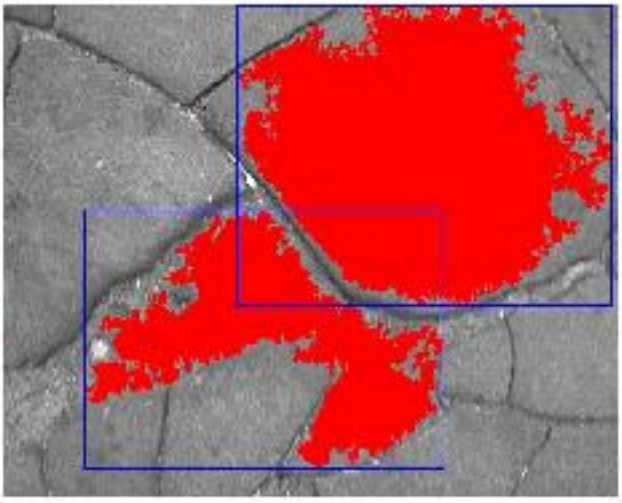
Here is another test case, with the actual pothole above , the image to the left was my initial test case, while the image to the right was with some changes and adding a centroid. I was able to detect the pothole a little bit more but here the program is a lot more sensitive and is detective smaller cracks that i deem necessary. It would be important to add function to remove centroids below a certain size and the team next semester should work on that. Also i have noticed that is there isn't enough of a distinct connection it will create esperate boxes, the pothole above is one but there is a noticeable gap between the two hole which is why the two largest centroids are quite separated.



As the program evolved the potholes above most latest results , as you can see it does identify the pothole region and draws the centroid that fits that shape



Above is another piece of testing that had some success , it could identify the cracks you see in the picture to the left but there is some information missing,



Above is a test case that ended in failure, the image to the left is the original and you can notice the potholes but when ran through the algorithm it thinks there are two potholes and thinks the right is a pothole. It is a strange error because the pothole and the surface have completely different intensities and it is quite visible from the picture how different the regios are. The test cases above had potholes that didn't looks all that different from the ground surface.

6.2.15 Final Design Review Comments/Reflection

- Delete unnecessary files as much/early as possible to eliminate big data issue
- Possible to use a partim of Otsu's method to detect likelihood of pothole to determine quick first step on frame importance?
- For plane setting, is there an accuracy setting? Or does it only consider changes in mm?
- How often do you need to collect data to be effective? How long does it take to move from one view frame to the next? Maybe use GPS to determine distance change?
- Break project down into phases that can be delivered.
- How does plane fitting react to differences in road or changes in camera height?
- Would be nice to see pothole image alongside heat map and binary.
- How long would it take to process one day of data; are multiple processors needed? Can you filter and only analyze frames that have a pothole? (<1fps analysis rate)
- More clarity in decision making process: add more clear weighted decision matrices.
- How do you account for the motion during impacting a pothole and variable surfaces?

6.2.16 End-of-Semester Summary

We have covered the following this semester:

- Commenting of code in the file dataAnalysis17.py: We significantly update the code, and have it in the "master" branch of the github repository. You can clone the repo as would be explained in our transition document.
- We corrected a few bugs in plane fitting algorithm, but many remain.
  Suggestion: You will see that plotting the graphs takes a lot of time. This is probably because the number of pixels is very high. You can try and decrease the density of pixels (probably 1:16 downsize of the pixels). This will improve the time, leading to easier, faster debugging!
- We have designed a rough algorithm for reading and nomenclature of files to be obtained from Hardware team. This can be implemented. It can be found in Kalpan's notebook, "Works and Accomplishment".
- We have also experimented with Image Tracking and there has been a code in Matlab which does image tracking.

6.3 Website and Application Development

The analyzes data from the DA team is given in location and severity of the pothole. The WA team displays that data on a website accessible to the city displaying the location and severity of potholes on a map. The WA team also develops a smart-device application accessible to the community/public, allowing people to report issues throughout the city, relaying that information back to the city, also displayed on the website map.

6.3.1 Existing Solutions

There are a few mobile apps that are available in the market which detects potholes. For instance, Street Bump[31], an application which residents of Boston can use to collect road condition data while they drive. It can detect bumps on the streets and provide the city with real-time information to fix short-term problems and plan long-term investments. However, this type of applications is city-specific and is not suitable for the city of West Lafayette. Other solution would be what the city of West Lafayette currently does which is manually assessing damages on the roads. In order to benchmark our potential solution, we could use the similar applications available in other cities (Street Bump, Get It Done San Diego Official[32]). Based on the initial benchmarking with the project partner, we've identified some needs and functionalities for the smartphone application:

User Need	*Specification*
City must be able to easily see frequency of pothole reports	• Application must have an admin login option, with a password requirement (backend) • Must have a map to show frequency of reports to admin users • Must have a map to visually plot reports, as well as an address description of the general area, instead of exact coordinates • Map must merge submissions in the same location as hardware data
City must be able to determine the severity of the potholes	• Ability to take picture of damage • Users must be able to upload pictures of road damage • Users must be able to write a description of the damage • Users must be able to document pothole severity in their submission • Examples of severity should be provided to users to assist in the rating of severity • Ability to send the location of damage
Application must be accessible and appealing to the widest range of users possible	• Application should be programmed for iOS • Application should be programmed for Android • Option to make submissions anonymously • Application should require less than 1 minute to submit a report of damage

We discussed many strengths and weaknesses that would allow for convenience and speed for user-application interaction. We wanted to identify functions that would provide more accurate and more useful information to the community partner. After an initial meeting with our community partner, we assigned roles to create a conceptual design that would "provide the best user ability and practicality to all of our stakeholders" (Marcus, assistant city manager). Various conceptual designs that our team created:

[31] http://www.streetbump.org/about
[32] https://www.sandiego.gov/get-it-done

Figure: Conceptual designs of Smart City Cities App

List of questions and criteria for project partner meeting, assumed needs to be met:

Question	**Response**
What data do you want to be able to collect? ● Size of pothole (SML)? ● Description box? ● Do you need a picture? ● What needs to be in the picture?	● Picture would be nice ● But worried about asking for a picture ● Maybe can be option, but not necessary because of safety ● App would be more for street department ● Would be developed for immediate identification ● Better to have a map than location services because of location of person ● Like point on the area ● Maybe both??
How do you want to collect location data? ● Is GPS at location of phone enough? ● Specific address?	● Location of pothole ● Coordinates
What kind of user data will be useful to you? ● Name is enough? ● Anonymous is allowed? ● Maybe a Facebook/Gmail sign up?	● Maybe jerks are putting random points… ● Not really needed to sign in, can be anonymous
Is the app meant to be long term? ● Does it only need to focus on potholes, or cracks and other things as well? ● maybe can be used for broken street lamps, broken stop signs etc.	● only potholes ● Maybe leaves? ● But not most important
Do you want other people to see all reports made on a map? ● Avoiding repetitive reports ● Confidential? ● admin privileges to report when a report has been resolved	● Want to see the same pothole reported more than once to show importance ● Don’t show other reports because will compromise showing frequency ● Admin privileges would be good
iOS and Android?	● Whatever is easiest, look at statistics ● Preferably both ● Want to get as many people as possible

Spring 2018 continued where Fall 2017 left off with the same specifications from previous semesters. To be thorough, we asked the project partner once again regarding goals and specifications desired within the application. The following are the questions that were reiterated:

Question	**Answer**
Specifically, who will be utilizing the app? Will the app be available to the public? Or only for city use? Is there a specific design that is preferred for the application?	● No, you can design what you think is best. ● Simple, clean, and professional design so that residents who are using the app aren't confused and can quickly figure out how to use it.

What specific features are you looking for in the app?	● Giving residents the ability to report issues (potholes, drainage issues, broken sidewalks, missing or damaged street signs, traffic concerns, and other issues)
	● Giving us enough information to find the issue and fix it quickly such as description, location, and an image (optional).
	● Currently residents can either call or email us, or submit a complaint via our website (https://www.westlafayette.in.gov/egov/apps/action/center.egov?view=form;page=1;id=126)
	  ● Giving residents another option for reporting complaints will be beneficial and may reduce the number of calls coming into our office. That would be the base desire of the app. Another use for the app could be to include a notification or information sharing system so that we could send an alert to people when a road is closed, when we are doing construction, if roads are icy, etc. Right now we use Nixle, which texts and emails people with that information, but residents might like another option.
	  ● Have the option to send an update once an issue has been addressed. Duke Energy uses a system to report street light issues. Once the light has been fixed, an alert is sent.

Specifically, who will be utilizing the app? Will the app be available to the public or only for city use?	● The general public to report concerns to us. ● Back-end website would only be available to city employees.

After collecting an initial project partner meeting, the team decided to draw a sketch to decide what we want our app to look like. We considered affordability, user needs, and all stakeholders as main functionality needs. We decided to design a simplistic front page to keep space for future additions. For affordability, we wanted to release the app to one app store only. This will help us test out the application first, and resolve any issues, before releasing it on another app store as well. With user needs, the app has to be easy to use, free of charge and it will process quick submissions. Finally, while brainstorming, we also had to make sure we took all the stakeholders into account.

After analyzing each conceptual design, the team has decided to stick with the following design (simplistic, modern, and practical):

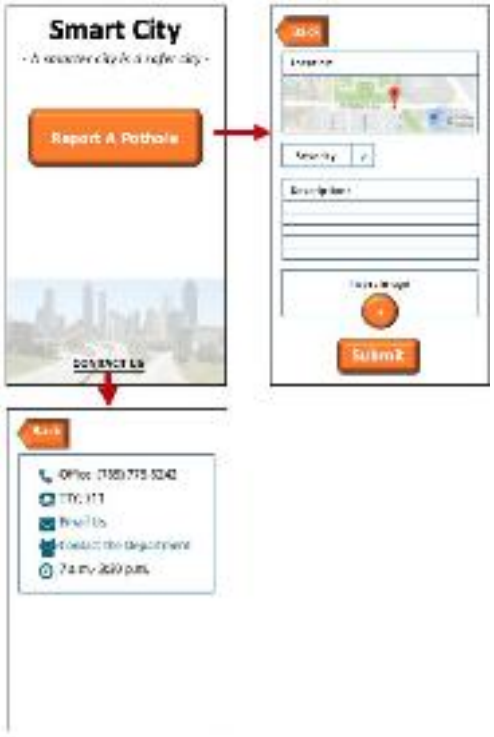


Figure: low-resolution prototype

From this concept, the team learned that the submit button functions as a data sender to the Purdue data storage server at EPICS. Also, we learned that the severity button should be a dropdown and not a pop up – simplifying the application functions with little possible error. We also learned that certain images are copyrighted and cannot be used in the platform (we took our own pictures). Overall, the team was excited to see that the functional prototype produced little errors while using the application.

**Note*: The chosen design opens up to a very simplistic and straightforward screen, where the name of the application is highlighted, with an option to report a pothole or contact the city (leading the user into different pages). The opening screen includes report and contact to increase user-friendliness and allow users to complete their objective in an efficient manner. The "Report A Pothole" button is very eye-catching, and takes users to a second page. Users will be able to confirm the location of the pothole on a map, select the severity of the pothole from a preselected list of severities, describe the pothole with a 2-3 description, or add a picture of the pothole. Options to describe or add a picture of a pothole are not required, but users will need to put fill in one of them. After finishing up their report, users are able to submit the pothole report to the city. If the user accidentally got to pothole report page, instead of contact

the city page, there will be a "Back" button conveniently located on the top left corner for users to click. The "Contact Us" button is located on the bottom of the opening page that takes users to a page with information regarding contacting the city with city business hours. Finally if the user wishes to go back to the main page from the contact page, there is another "Back" button located in the same location as the other to reduce room for confusion.

Due to our project scope, we do not have a typical manufacturing or assembling process. Our end-deliver is a smart-device application and is not tangible – digital propitiatory property does not require physical manufacturing.

Github is an online programming repository for open source collaboration. Downloading Github and the programming compiler (translates human language to machine language) was long and difficult. Implementing Github[33] and compiler installation will be documented in our transition document. Also the code behind our process was "assembled" by our team. While this code will stand alone and not be changed following delivery of the product, it is still important to document what each part of this code does. The practicality of documenting commands and functions in a user manual type setting is non-existent. Instead the code which we will deliver will be commented thoroughly. Not only will this be beneficial upon the delivery of the product, but also in the transition between semesters. This documentation will be important in the increase of efficiency between semesters.

Here are bill-of-materials that may be necessary to understanding/implementing future progress:

Item	**Made/ Bought**	**Vendor**	**Quantity**
Homebrew	Downloaded	https://brew.sh/	1
Node	Downloaded	React Native	1
Watchman	Downloaded	React Native	1
React Native	Downloaded	Terminal; instructions: https://facebook.github.io/react-native/docs/getting-started.html	1
Xcode	Downloaded	App Store	1
Android Studio	Downloaded	Google Play	1
Github	Downloaded	https://github.com	1
Sublime Text	Downloaded	https://sublimetext.com/2	1
Notepad ++	Downloaded	https://notepad-plus-plus.org/download/v7.3.3.html	1

Table: Bill-of-materials for future progress

The team has already faced several setbacks resulting in delays. One example of this is the difficulty we experienced while downloading the selected compiler. Spring 2017 App was not able to complete the application code prior to the end of this semester. Some of our time constraints include spring break and individual time allocated for studying and other class work. Our EPICS document deadlines and individual time allocated for studying and exams. There are no external time constraints imposed by our project partner. While they have indicated that a timely delivery of this phone application is favorable there is no required publication date.

**Note:* Submit functionality currently shows errors and we suggest looking through where objects are being called and changed within the React Native environment.

[33] Contact Kartik at mittal38@purdue.edu for access to Github

Note*: Our detailed design did not implement data saving (server, locally, writing data to .txt). Generally, App needs to implement the back-end analytics and we suggest collaborating with **6.2 Data Analysis.

For Fall 2017, strides were made in making an actual functioning app. The code was scrapped from the previous semester, as it allowed for us to have more freedom in creating what we have envisioned. One of our early prototypes for the application ended up looking too bulky, so we decided against using it:

One of our problems with the option above was that it took too many screens to get to reporting. We have opted for a single screen experience, going for a look similar to that of Uber:

This gave Fall 2017 the functionality Spring 2017 semester was looking for (ability to take pictures, to report severity, and to use GPS), while also incorporating a sleek, efficient overall layout. The design has a permanent marker in the middle that would allow for the user to simply place where the pothole was that they saw, and can convert it to an address later for the city using the Google Maps API. All the needed features are on the one page, and next to the address bar on the top of the image there is an 'options' button that will give contact information for the city. Fall 2017 intended to add a 'severity' scrollbar, but due to time constraints, were unable to execute. Thus, this has become a goal for the Spring 2018 semester - a 'severity' scrollbar that has only 'mild', 'moderate', or 'severe' to simplify and categorize the potholes for people deciding on the severity of the pothole. These options are much more concrete than a number scale from one to ten. Each pothole will still have a ranking for the city to determine which pothole requires more attention than others.

Code was developed for what this will look like, and we currently have a layout prototype with some functionality. To make things easier for the city, we have created a website for the reported data to get sent to so that it can be seen without a problem for city administrators, but cannot be viewed by the general public. Below is a view of our current app, with its current functionality, and our near fully operational website:

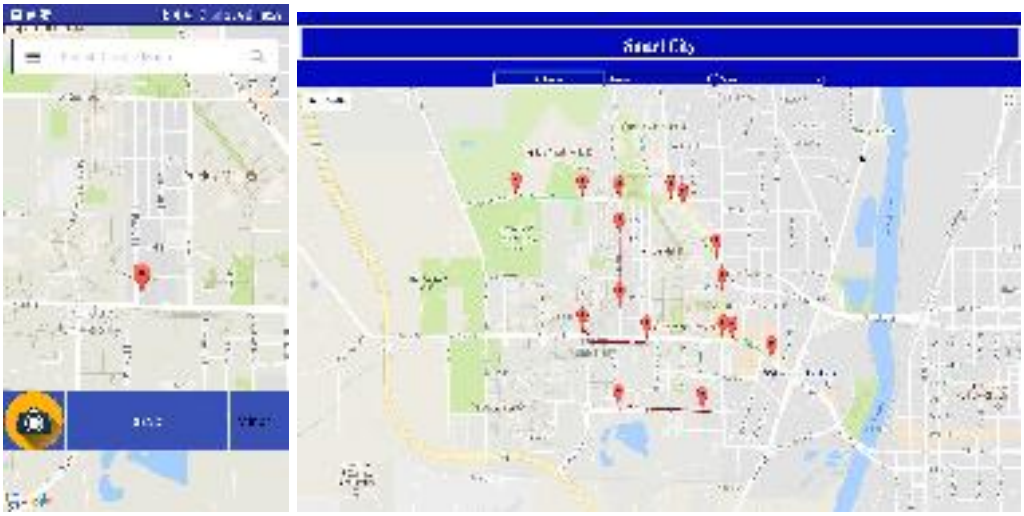


Figure: A view of the app and website

Figure: Enhanced version of the app (Spring 2018)

We are continuing to add functionality, but for now we have attained a scrolling map with a working camera option. Soon we will link our app and website to get reported points to show up on the site. Already, the website is able to calculate severity of roads and can filter through multiple options. Spring 2018 is also making it a priority to consistently document code to ensure ease of transition between semesters as a result of personal experience. For this reason we used GitHub, which is a web-based hosting service for version control and storing the changes made by user is the code. It also gives us the option to revert back to anhy old code so if we hit a dead end, we can go back to a point where we want. It is mostly used for computer code. It offers all of the distributed version control and source code management functionality of Git as well as adding its own features

The prototype of the app does have the functionalities that we intended to include within it; however, the layout does not look like the way that it is supposed to be. The bottom bar does not stick to the bottom and more stays in the middle, which distracts the view of the map. Thus, Spring 2018 App Team fixed this problem after learning the code process of Android Studio and the result is shown as the figure above. Spring 2018 is also trying to come up with the ways to improve the user interface to increase the accessibility of the app for the users.

The main objective for Spring 2018 semester was to have a backend server for the app where the information of pothole like coordinates, severity, image encoded in Base64 format (this is an encryption format which convert an image to string using inbuilt Android libraries so that it is easy to store on the server and decrypt back to an image to display it on the website) and time stamp as in what time the data was sent to the server. The team brainstormed on what server to choose and made a decision matrix included below to make it more clear which one to use. The 3 different option we narrowed down for server were - Google’s Firebase, Amazon AWS Mobile Hub and Microsoft’s Azure.

What is Firebase?	What is AWS Mobile Hub?	What is Microsoft Azure?
Firebase is a cloud service designed to power real-time, collaborative applications. Simply add the Firebase library to user application to gain access to a shared data structure; any changes user makes to that data are automatically synchronized with the Firebase cloud and with other clients within milliseconds.	AWS Mobile Hub is the fastest way to build mobile apps powered by AWS. It lets users easily add and configure features for their apps, including user authentication, data storage, backend logic, push notifications, content delivery, and analytics. After users build their apps, AWS Mobile Hub gives them easy access to testing on real devices, as well as analytics dashboards to track usage of their apps – all from a single, integrated console.	Azure is an open and flexible cloud platform that enables user to quickly build, deploy and manage applications across a global network of Microsoft-managed datacenters. User can build applications using any language, tool or framework. In addition user can integrate his or her own public cloud applications with existing IT environment.

After thinking it through, we made a decision matrix to choose the most optimal option.

Factors: (0-5 scale)	Cost	Quality	Reliability	Accessibility	Total
Google's Firebase	4	5	5	5	**19**
Amazon AWS	3	4	5	3	**15**
Microsoft's Azure	5	5	5	3	**18**

Benchmark

	5	4	3	2	1
Cost	< $25	$25~$34	$35~$44	$45~$54	≥ $55
Quality	≥ 100k simultaneous connections	100k~1k simultaneous connections	1k~500 simultaneous connections	500~100 simultaneous connections	< 100 simultaneous connections
Reliability	Excellent security	High security	Average security	Low security	No security
Accessibility	Fully compatible with Android	Almost compatible with Android	Partially compatible with Android	Barely compatible with Android	Not compatible with Android

Figure: Data stored on Google's Firebase server

The image above shows how actually the data is stored in the server. The information is stored in JSON file (JavaScript Object Notation file) on the server which is easy to read and understand by humans as well as softwares.

Overall, Spring 2018 is focusing on implementing the expected features as well as additional ones that have been requested by the project partner.

6.3.2 Project Specifications (Spring 2018)

Website

- ❏ Website for the city to review this information
- ❏ Pictures can be used to identify potholes (deep learning)
- ❏ Place for city to be able to view reported pictures
  - ❏ Attached with markers in website
  - ❏ Streets with many potholes are painted based on their severity
- ❏ Read data from a server
- ❏ Usage of simple GET requests
- ❏ Use information gathered by both the users of the app and the POLES team

- ❏ Adjust data points to correct for curvature in streets
- ❏ Use of express, bootstrap and jquery to make web development easier
- ❏ Online programming environment for the website that will later be transferred into a server (cloudnine)
- ❏ Website will only be accessible through the city's network
- ❏ Local password might be instated (if the city needs one)
- ❏ Easy to use UI
- ❏ Visually appealing styling (simple styling)
- ❏ Filtering of the potholes based on Severity, street and date (month)
- ❏ Programmed to work on all screen sizes including mobile

API

- ❏ Map API integrated into app
    - ❏ Map can move around
    - ❏ Move point in center of map (e.g. Uber)
- ❏ Address bar able to locate positions
    - ❏ Street name, city name, state, zip code
- ❏ Location Services
    - ❏ Current location
    - ❏ Location of the address typed
- ❏ Camera API integrated into app
- ❏ Usage of simple POST requests
- ❏ Developed to run on both new and old android version
    - ❏ Developed UI

User Experience

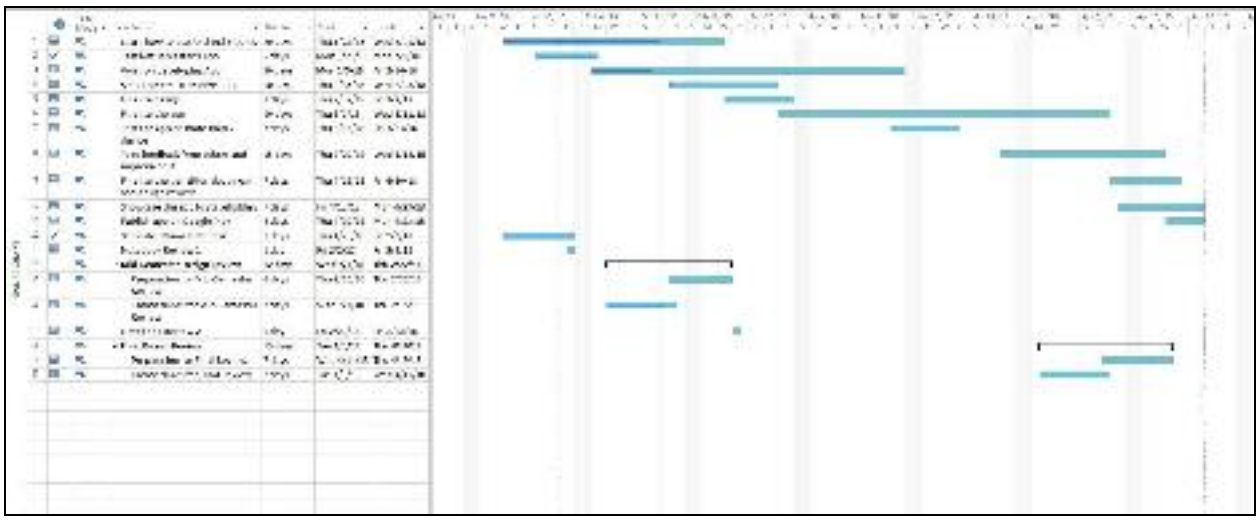
- ❏ Scrollbar for severity
    - ❏ Use of three severity levels:
        - ❏ Mild
        - ❏ Moderate
        - ❏ Severe
- ❏ One screen layout, easy to use even for first time users
    - ❏ Map functionality easy to interact with
- ❏ Visually appealing interface
- ❏ User safety accounted for
    - ❏ Alert/notification system

Data Collection

- ❏ Setting up an appropriate server
    - ❏ Enough space to store data
    - ❏ Compatible with both app and website

- ❏ Efficient data package
  - ❏ Stores picture separately from text data
  - ❏ Packages data to be sent to server for storage
- ❏ Take user input into data package
- ❏ Send data to server

Overall Timeline



6.3.3 Website Progression: Spring 2018

The team from previous semester had done a fantastic job on the website. The website is functioning with basic functions (filter by severity or date, marking on the map, and hide/unhide data collected by application or kinect). Our goal for Spring 2018 was to host the website so that we can see if it is actually working properly on the internet as it is on the local host. We tried to use Firebase but it does not work. Apparently, we found out that the website was written in JavaScript with node.js . If you do not know what is node.js and you are working on the website, I recommend you to learn about it ASAP.

The code can be found on the team's SharePoint, or in the Google Drive folder[34]. The code for the website is written on a cloud environment called Cloud9 (https://aws.amazon.com/cloud9/) which is preferable just in case the code is lost in your personal computer. If you have never used Cloud9 before, we have made a manual on how to use it which can be found on one of our members' notebook (you may need to login). The explanation of what those files do are also attached in the notebook. If you have any further question on how to run the website, feel free to contact one of us.

For the moment, the website is only accessible when a free temporary server is ran. This service is provided by Amazon's Cloud9. However, this server cannot support a high traffic data transfer and it will automatically shut down if left idle for more than 15 minutes. Fortunately, this is good enough for the team as we are only trying the prototype. If this website needs to be published, it needs to be transferred into a more proper server.

[34] https://drive.google.com/drive/folders/1uDLePJS4xR6xgMzAiFkGTSvftO1iwooU

Hosting

For Spring 2018, we tried to host the website using Google Firebase hosting service with the assumption that it would be easier to link it with the database as it is using Firebase too. However, the map does not show up on the link (https://epicssmartcity.firebaseapp.com). We figured maybe because the website was written in JavaScript (JS) while Firebase can only host HTML. So, we made an attempt to convert the JavaScript to HTML. Unfortunately that does not solve the problem. Advisors suggested us to meet Jason Dufair[35], a senior software engineer working at Purdue Learning and Teaching Technology to help us in this issue.

We met Mr. Dufair and he explained everything that needed to be done to host the website as it was written in JavaScript. Later, we did what Mr. Dufair told us and this does not work too. Then, we found out that the way Cloud9 work is a bit different than others. We met a few of our friends from Computer Science major but they said that this environment (Cloud9) works a bit differently. Then, our teaching assistant advised us to contact Google Firebase Support to help us with this issue. Here is their response:

> Thanks for reaching out! Unfortunately Firebase hosting currently doesn't support server side scripting languages so Node.js files won't run when you deploy them to Firebase.
> We are looking into supporting server side scripting in the future but I don't have any timelines to provide at the moment. Be sure to keep an eye on our release notes for any further updates.
>
> You can also check our Cloud Functions and see if it meets your use case. You will be able to connect the functions with Firebase hosting as well.
>
> Let me know if you have any questions.
>
> Thanks,
> Wiley

This contradicts to what Mr. Dufair said that Firebase should be able to host it. He is not wrong. The Firebase does support node.js, only that it does not support Cloud9. Therefore, we looked into a few others alternatives that can support node.js hosting. Below some of the hosting service that we found:

1. OpenShift (https://www.openshift.com/)
2. Nodejitsu (https://www.nodejitsu.com/)
3. Microsoft Azure (https://azure.microsoft.com/)

However, please do acknowledge that if you are using Cloud9, make sure that the hosting you are using can support the integration of Cloud9.

The main issue here is using Firebase on Cloud9. There is not much information and reference to do it. You will find a lot of websites and videos tutorial on hosting node.js on the internet. That was what we did and usually it was just find at the beginning. However, at some point it would not be the same as the video anymore. This is where we think you will need someone with advanced coding skill especially JavaScript.

[35] Jason Dufair is now with Learning and Teaching Technology so he is a bit difficult to be reached in person. He was a staff in EPICS and helped many students about software relating problem. Contact : jase@purdue.edu

We believe that there is a way to work around this. Only that none of us knows how to do it. The progress of this website will require someone who has experience working with node.js

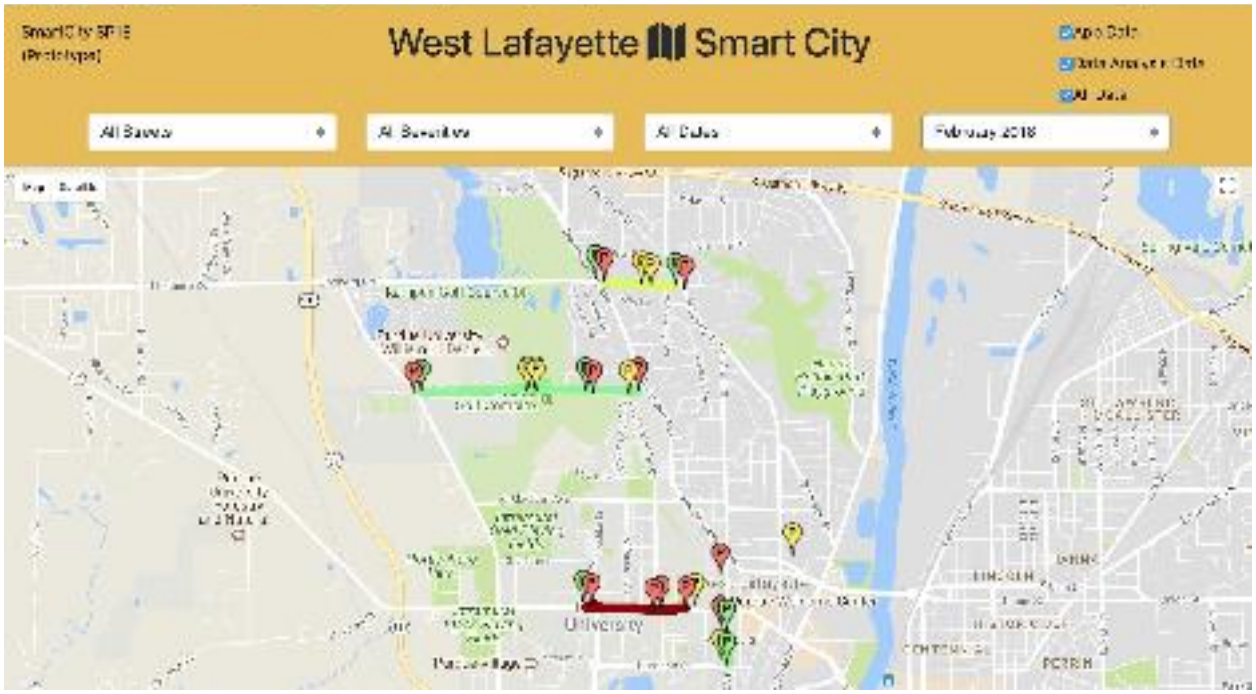
Connecting to a Server

For now, our Android application has been successfully integrated with Firebase Database server. More information can be seen in the application section.

We consulted Mr. Dufair about integrating the website with the data collected on the database. He said it should be easy as we only need to add a few lines of code and the APIKEY, etc. We have not tried to connect it with a server as we only connected the application with database quiet late in the semester and we were trying to figure out how to host node.js

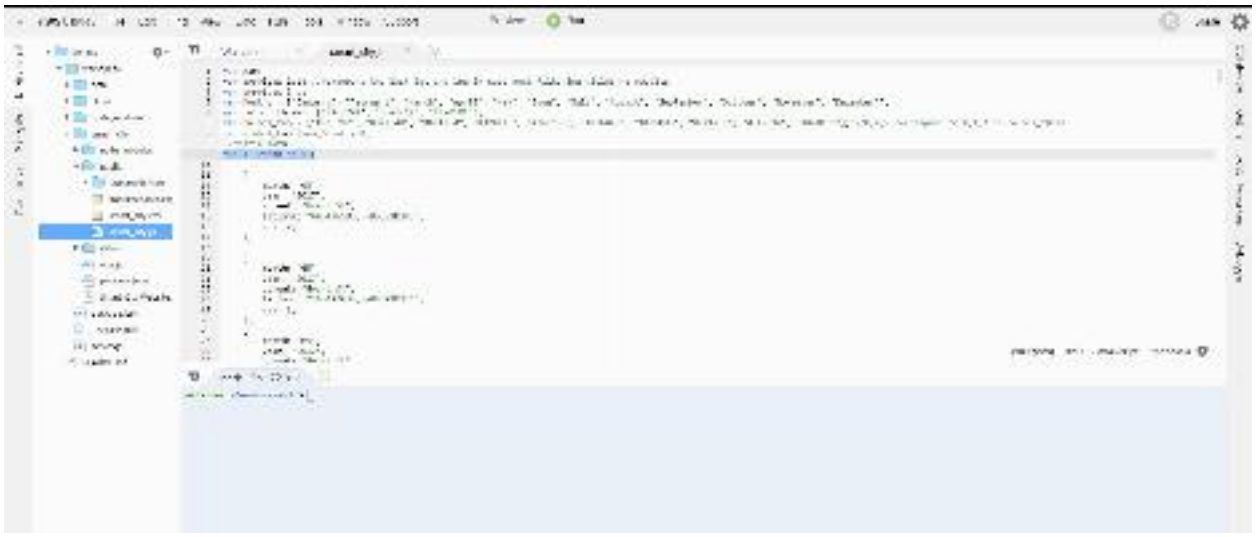
Data (Markers on Map)

The Spring 2018 team learned that all the data in the map (below) are hardcoded. This is the only option available as no server is established yet. The server is needed so that the data from Kinect and application can be stored in one place. The plan is to make the website read the data from the server and automatically update it on the location instead of manually entering the code. The current focus of the website is to display data from both app and kinect as pothole reporting website is already available on the official City of West Lafayette's website.



The code for the markers can be found at workspace > smart_city > public > smart_city.js . **var allPotholes** is the code where all the data for “App data” filter while **var allPotholes_DA** is where the data for “Data Analysis” filter.

Also attached is the location of the marker, and the line of the code (assuming the code is untouched):



6.3.4 Final Design Review Comments/Reflection

- All decision matrices need to be weighted and ranges specified.
- Include “usability;” ease for a user to work with a server service.
- How do you differentiate between levels of severity from different users?
- Implement feature so that the app cannot be used while driving.
- Will there be any type of prioritization when scheduling repairs? Will traffic data be incorporated?
    - Future implementation of re-routing feature
- Address why the app is only being designed for Android at the moment in the Design Review.
- How were coding and interfacing decisions made, and does this meet the needs of majority of users?
    - Reiterate the needs of the project partner.
    - Include decision matrices.
- Data flow diagram was over simplified and confusing.
- Address security and data ownership on server selection.
- Size constraints and/or optimization in app before sending through app?
- Consider how the website will be monitored and updated when potholes are fixed.
- What happens if two people mark the same pothole on the app? Does the severity level update with the most recent data received?

6.3.5 End-of-Semester Summary

For Spring 2018, we continued what we had from previous semester and made much progress in improvements as well as additions. This semester, we particularly focused on having proper documentation for our team so that the future semester don't have a problems understanding what has previously been completed. Moreover, for the app, we included a RealTime Database server so that we have the information collected from the user of the reported pothole. This was a major accomplishment as any app needs a server to collect data. We also fixed the user interface of the app so that it appears robust. We also did some error handling in the app such that the user cannot send the information unless they have specified the severity and taken a photo so that inaccurate information is sent. Additionally, a major feature we added is to use the GPS of the device to get user location so that the user starts the app with the map pointing to their current location. This makes it easy for the user to start off the app with their own location.

For future semesters, we hope to do rigorous testing of the app and deliver it to general users to get feedback and suggestions for improvements regarding, but not exclusively, usability. Moreover, as this app is current only compatible with Android devices, futures semester should have a goal of delivering an iOS version as well. Additional features should also be implemented to report general issues apart from potholes as per the project partners' need.

Appendix A: Past Semester Archive

A.1: Team Members

A.1.1: Fall 2017

Name	*Role*
Mohammad Jahanshahi	● EPICS Advisor o Advises EPICS Syllabus learning objectives o Guidance in sensor technology/analysis o Initial project innovator
Margaret Phillips	● EPICS Advisor o Advises EPICS Syllabus learning objectives o Guidance in academic research and group cohesion/leadership
Dahjung Chung	● Teaching Assistant o Academic logistics and operations for EPICS section o Guidance in sensor technology/analysis
Eric Jin Wook Choi	● Project Manager – responsible for overall operation and effectiveness of team and provides planning, direction, and guidance ● POLES (Data Analysis) o Detection and quantification of potholes o GUI (front-end) development
Mohammad Kobeissi	● App Design Lead o Oversees App design o Responsible for facilitating project through components of design process o Responsible for project planning, execution, risk assessment to deliver a quality team end-deliverable on time/budget
Fajar Ausri	● POLES Co-Design Lead o Oversees POLES design o Responsible for facilitating project through components of design process o Responsible for project planning, execution, risk assessment to deliver a quality team end-deliverable on time/budget ● POLES (Data Analysis) o Ensuring Kinect data collection (data management, error analysis) o Quantification of potholes o Google Maps API implementation to GUI (front-end)

Grant Hilbert	● POLES Co-Design Lead o Oversees POLES design o Responsible for facilitating project through components of design process o Responsible for project planning, execution, risk assessment to deliver a quality team end-deliverable on time/budget ● POLES (ultrasonic) – ensuring ultrasonic data collection (data management, programming)
Gytis Kriauciunas	● Financial Officer – develop/manage project's budget ● POLES (GPS) – ensuring GPS data collection (data management, programming)
Yvette Chowdry	● Project Archivist – ensuring quality of project documentation and documentation practices ● App – GitHub implementation
Tiyani Hu	● Project Partner Liaison o Communication between teams and the community project partner o Inform on regular basis of progress of the project and relevant team documentation for partner observation/comment ● POLES (Kinect) – ensuring Kinect data collection (data management, programming, error analysis)
Weili Wang	● Webmaster – update/maintain project's website ● POLES (GPS) – ensuring GPS data collection (data management, programming)
Lexie Plocher	● POLES – ensuring inter-team communication, group dynamic, and team milestones
Nicholas Briggs	● App – pothole reporting functionality
Nicholas Idso	● App – GUI (front-end) development

A.1.2: Spring 2017

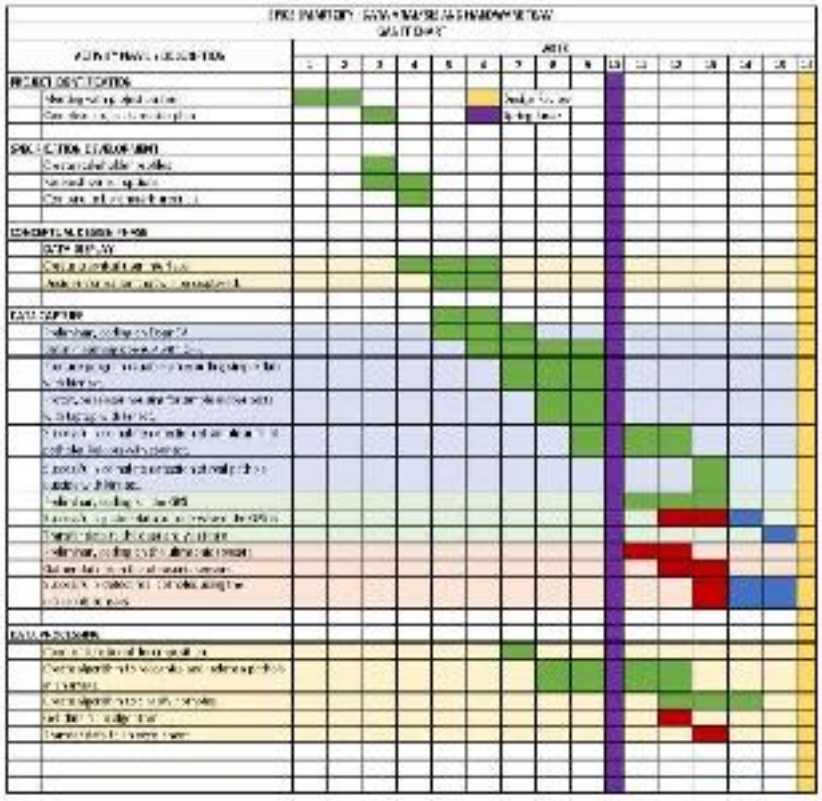
Name	*Role*
Mohammad Jahanshahi	● EPICS Advisor ○ Advises EPICS Syllabus learning objectives ○ Guidance in sensor technology/analysis ○ Initial project innovator

Margaret Phillips	● EPICS Advisor ○ Advises EPICS Syllabus learning objectives ○ Guidance in academic research and group cohesion/leadership
Dahjung Chung	● Teaching Assistant ○ Academic logistics and operations for EPICS section ○ Guidance in sensor technology/analysis
Eric Jin Wook Choi	● Project Manager – responsible for overall operation and effectiveness of team and provides planning, direction, and guidance ● POLES (Data Analysis) ○ Detection and quantification of potholes ○ GUI (front-end) development
Mohammad Kobeissi	● App Design Lead ○ Oversees App design ○ Responsible for facilitating project through components of design process ○ Responsible for project planning, execution, risk assessment to deliver a quality team end-deliverable on time/budget
Fajar Ausri	● POLES Co-Design Lead ○ Oversees POLES design ○ Responsible for facilitating project through components of design process ○ Responsible for project planning, execution, risk assessment to deliver a quality team end-deliverable on time/budget ● POLES (Data Analysis) ○ Ensuring Kinect data collection (data management, error analysis) ○ Quantification of potholes ○ Google Maps API implementation to GUI (front-end)
Grant Hilbert	● POLES Co-Design Lead ○ Oversees POLES design ○ Responsible for facilitating project through components of design process ○ Responsible for project planning, execution, risk assessment to deliver a quality team end-deliverable on time/budget ● POLES (ultrasonic) – ensuring ultrasonic data collection (data management, programming)
Gytis Kriauciunas	● Financial Officer – develop/manage project's budget ● POLES (GPS) – ensuring GPS data collection (data management, programming)

Yvette Chowdry	● Project Archivist – ensuring quality of project documentation and documentation practices ● App – GitHub implementation
Tiyani Hu	● Project Partner Liaison ○ Communication between teams and the community project partner ○ Inform on regular basis of progress of the project and relevant team documentation for partner observation/comment ● POLES (Kinect) – ensuring Kinect data collection (data management, programming, error analysis)
Weili Wang	● Webmaster – update/maintain project's website ● POLES (GPS) – ensuring GPS data collection (data management, programming)
Lexie Plocher	● POLES – ensuring inter-team communication, group dynamic, and team milestones
Nicholas Briggs	● App – pothole reporting functionality

A.2: Fall 2017 Timelines

A.2.1: Data Analysis and Hardware Team Fall 2017



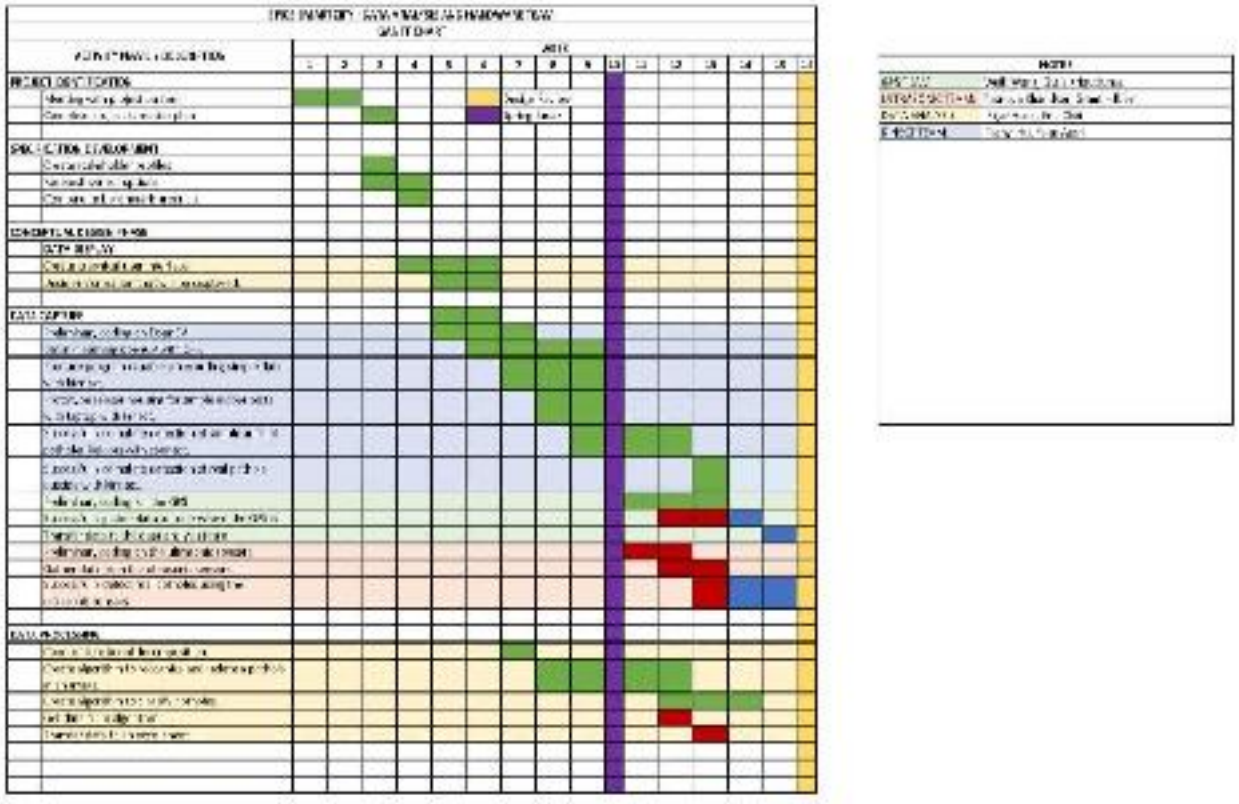
A.2.2: App Team

Phases	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Project Identification Phase																
Description of the Community																
Stakeholders																
Social Context																
User Needs																
Specification Development Phase																
Benchmarking/IP																
Specifications																
Conceptual Design Phase																
Brainstorm																
Low Resolution Prototyping																
Concept Convergence																
Proof-of-Concept Prototyping																
Proposed Solution																
Detailed Design Phase																
Design Process																

B.O.M.s																
Manufacturing/Assembly Process																
Risk Analysis																
Verification																
Validation																
Delivery Phase																
User Manual																
Waiver Release & Hold Harmless																
Customer Satisfaction Questionnaire																
Delivery Checklist																
Approvals																
Service/Maintenance Phase																
Retirement or Redesign																

A.3: Spring 2017 Timelines

A.3.1: Data Analysis and Hardware Team



A.3.2: App Team

	Week															
Phases	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Project Identification Phase																
Description of the Community																
Stakeholders																
Social Context																
User Needs																
Specification																

Development Phase																
Benchmarking/ IP																
Specifications																
Conceptual Design Phase																
Brainstorm																
Low Resolution Prototyping																
Concept Convergence																
Proof-of-Concept Prototyping																
Proposed Solution																
Detailed Design Phase																
Design Process																
B.O.M.s																
Manufacturing/ Assembly Process																
Risk Analysis																
Verification																
Validation																
Delivery Phase																
User Manual																
Waiver Release & Hold Harmless																
Customer Satisfaction Questionnaire																
Delivery Checklist																
Approvals																
Service/Maintenance Phase																
Retirement or Redesign																

Appendix B: Overall Project Design

Smart City Cities developed its own design process/timeline apart from the expected EPICS framework. For current progress timeline, see **5.4 Semester Timeline**. The following descriptions are provided from the Smart City Appendix[36] and detailed status/evidence is provided.

B.1 Project Identification

Description: Each document is should include a *Project Identification* outlining and describing the specific problem that the *team* will be addressing and some preliminary description of the overall function of the end-product (which preliminary needs will be fulfilled, describe what the solution/product will do, and how the solution/product will solve the preliminary problem/need). Descriptions should be based on some educated support (secondary research, gauging small sample of the population, peer-reviewed sources/citations, etc.). Explain why the team's problem is worth addressing (socioeconomic or geopolitical impact).

Provide a description of Smart City's community project partner and some of the needs (portray how *your team* views and identifies the partner and their needs). Identify primary/secondary users and stakeholders (description and needs). Project identification of users/stakeholders and needs should be preliminary description based on initial meetings with the project partner and initial need-finding of a small sample of users. Further elaboration should be done in **Overall Project Design** > *Functional Evaluation*.

**Note*: Goal is to identify a specific, compelling need to be addressed

Phase 1: Project Identification	**Status**	**Evidence can be found:**
Conduct needs assessment (if need not already defined)	**Status:** *Completed* **Semester:** *Spring 2017*	• **4.1 Project Charter** • **4.4.1 POLES Outcomes/Deliverables** • **4.4.2 Smart City City App Outcomes/Deliverables**
Identify stakeholders (customer, users, person maintaining project, etc.)	**Status:** *Completed* **Semester:** *Spring 2017*	• **4.2 Stakeholders**
Understand the Social Context	**Status:** *Completed* **Semester:** *Spring 2017*	• **4.4.1 POLES Outcomes/Deliverables** • **4.4.2 Smart City City App Outcomes/Deliverables**
Define basic stakeholder requirements (objectives or goals of projects and constraints)	**Status:** *Completed* **Semester:** *Spring 2017*	• **4.1 Project Charter**
Determine time constraints of the project	**Status:** *Completed* **Semester:** *Spring 2017*	• **4.4.1 POLES Outcomes/Deliverables** • **4.4.2 Smart City City App Outcomes/Deliverables**

[36] Smart City SharePoint>Spring2017>Smart City Appendix

B.2 Specification Development

Description: Prepare a background review based on initial project partner engagement and conceptual design brainstorming relevant to the users and stakeholders. Discuss requirements and limitations of a prototype based on users/stakeholders, technology, and environmental issues (identify and define the design issues associated with the prototype). Reference any peer reviewed documents to support the team's decisions (refer to Purdue Libraries). Discuss the performance requirements/limitations of the prototype (consider the level of needed functionality of a product and how the product will perform various functions to satisfy needs and activities). Refer to the **Semester Team Information** discussion of cost-analysis breakdown and consider the effects of cost into the performance of a prototype/end-product. Provide some discussion to the possible inter-project limitations that may influence the initial prototype and field testing (waiting on another team to finish a progress, communicating/establishing testing standards, differing user/stakeholder testing targets, etc.).

**Note*: Goal is to understand "what" is needed by understanding the context, stakeholders, requirements of the project, and why current solutions don't meet need, and to develop measurable criteria in which design concepts can be evaluated.

Phase 2: Specification Development	**Status:**	**Evidence can be found:**
Understand and describe context (current situation and environment)	**Status:** *Completed* **Semester:** *Spring 2017*	● **4.4.1 POLES Outcomes/Deliverables** ● **4.4.2 Smart City City App Outcomes/Deliverables**
Create stakeholder profiles	**Status:** *Completed* **Semester:** *Spring 2017*	● **4.2 Stakeholders**
Create mock-ups and simple prototypes: quick, low-cost, multiple cycles incorporating feedback	**Status:** *Completed* **Semester:** *Spring 2017*	● **6 Current Design**
Develop a task analysis and define how users will interact with project (user scenarios)	**Status:** *To be done* **Semester:**	
Identify other solutions to similar needs and identify benchmark products (prior art)	**Status:** *Completed* **Semester:** *Spring 2017*	● **6 Current Design**
Define customer requirements in more detail; get project partner approval	**Status:** *Completed* **Semester:** *Spring 2017*	● **6 Current Design**
Develop specifications document	**Status:** *Completed* **Semester:** *Spring 2017*	● **6 Current Design**
Establish evaluation criteria	**Status:** *To be done* **Semester:**	

B.3 Conceptual Design

Description: Create a system map that outlines the connections and functions of every operation and activity of an ideal, initial prototype. The system map should clearly show that each function satisfies some activity or need of the users/stakeholders (system map forces designers to archive need-satisfaction and includes the users/stakeholders to the design table). An example of an optimal system map can be presented in lab.

Develop a simple prototype that can be built in a few minutes and utilize the initial prototype as a basis model to further improve upon. Do not create a fully-somewhat functional prototype. Tap into the engineering imagination and imitate the functionality and end-goal standards of an end-product to the initial prototype. Document your initial prototype with pictures and a description of functions based on a system map.

Return to the community project partner and discover if the team's *Project Specification* is appropriate and satisfies the users' and stakeholders' needs. Investigate the user's interaction with the initial prototype and note the smallest interactions details. Question and gauge individual interactions as they may provide insight into a "problem" that the team may have overlooked or unpredicted. An initial functional evaluation with the user allows the design team to get a sense of how someone might interact with your initial prototype and may reveal suggestions into the *Functional Prototype*.

**Note*: Goal is to expand the design space to include as many solutions as possible. Evaluate different approaches and selecting "best" one to move forward. Exploring "how".

Phase 3: Conceptual Design	**Status:**	**Evidence can be found:**
Complete functional decomposition	**Status:** *Completed* **Semester:** *Spring 2017*	• **6 Current Design**
Brainstorm several possible solutions	**Status:** *Completed* **Semester:** *Spring 2017*	• **6 Current Design**
Prior Artifacts Research	**Status:** *Completed* **Semester:** *Spring 2017*	• **4.3 Project Objectives**
Create prototypes of multiple concepts, get feedback from users, refine specifications	**Status:** *Completed* **Semester:** *Spring 2017*	• **6 Current Design**
Evaluate feasibility of potential solutions (proof-of-concept prototypes)	**Status:** *Completed* **Semester:** *Spring 2018*	
Choose "best" solution	**Status:** *Completed* **Semester:** *Spring 2017*	• **6 Current Design**

B.4 Detailed design

Description: Develop a physical model for functional evaluation. Refer to the system map and initial functional evaluation with the project partner and develop a physical functioning prototype. Document with pictures and descriptions as appropriate. In order to test the interactive and usability functionality of the team's prototype, develop a prototype that integrates some level of activity that the users will be able to learn the objective of the team's end-product.

Ideally recruit 7 users for an acceptable functional evaluation. One suggestion of testing may be to allow testers interact with the prototype through self-exploration (let the testers play with it). The main objective of functional evaluation is to obtain an initial assessment of the "goodness" of the need-finding, system map, and design process. Sitting down to gauge a small sample of testers will allow engineers to uncover unforeseen problems not identified within the scope of functional decomposition. For a functional evaluation to yield effective results, try to remove any team's bias in asking questions and imagine being in the shoes of an "everyday" user. For documentation, write up a description of how the team conducted field trials (what did you ask the participants to do? how did you recruit testers? how many testers? etc.), some secondary research on demographic information, and data/conclusions from field trials.

After an "effective" *Functional Evaluation*, return to the drawing table and implement changes based on data/conclusions. Redesign the *Functional Prototype* (simplifying functions, aesthetic adjustments, new activities, etc.) and incorporate the redesign into a new functional prototype. A *Revised Prototype* should be ideal product that an engineer would like to propose to his/her superiors for further research/investment. A revised prototype should be able to communicate the appearance (what the team had in mind for a product; shape, size, weight, color, etc.) for the redesign.

Field testing differs from functional evaluation. Field testing determines if the executable functions of a prototype yields acceptable results (does the prototype work to produce data?). For documentation, write up a description of how the team conducted field trials (what did you ask the participants to do? how did you recruit testers? how many testers? etc.), some secondary research on demographic information, and data/conclusions from field trials. *Please recruit different testers*.

Go back to project prospectus and project specification and access how well your redesigned solution compares to the team's original goals for the project. Identify and document what improvements have been made (include some reasoning why improvements had to be made/ why "problem" was not identified) and suggest further improvements for future EPICS redesign (suggestions should be optimizations goals to streamline the existing design framework established in this team's design document). Describe what your team views as the "best" solution for the users/stakeholders. Clearly state what final improvements are needed and why they are essential to the success of the users/stakeholders based on the team's findings in functional evaluation and redesign.

**Note*: Goal is to design working prototype which meets functional specifications.

Phase 4: Detailed Design	**Status:**	**Evidence can be found:**
Bottom-Up Development of component designs	**Status:** *Completed* **Semester:** *Spring 2017*	• **6 Current Design**
Develop Design Specification for components	**Status:** *In-Progress* **Semester:** *Spring 2018*	• **6 Current Design**
Design/analysis/evaluation of project, sub-modules and/or components (freeze interfaces)	**Status:** *In-Progress* **Semester:** *Spring 2018*	• **6 Current Design**

Design for Failure Mode Analysis (DFMEA)	**Status:** *To be done* **Semester:**	
Prototyping of project, sub-modules and/or components	**Status:** *To be done* **Semester:**	
Field test prototype/usability testing	**Status:** *To be done* **Semester:**	

B.5 Delivery

Description:

**Note*: Goal is to refine detailed design so as to produce a product that is ready to be delivered! In addition, the goal is to develop user manuals and training materials.

Phase 5: Delivery	**Status:**	**Evidence can be found:**
Complete deliverable version of project including Bill of Materials	**Status:** *To be done* **Semester:**	
Complete usability and reliability testing	**Status:** *To be done* **Semester:**	
Complete user manuals/training material	**Status:** *To be done* **Semester:**	
Complete delivery review	**Status:** *To be done* **Semester:**	
Project Partner, Advisor, and EPICS Admin Approval	**Status:** *To be done* **Semester:**	

B.6 Service / Maintenance

Name of Servicer/Maintenances	Date	Service/Maintenance Done Notes