**CSE 550**

**Team 8**

**[Cameron Vincent, Charles Weiss, Makayla White, Stone Barrett, Lauren Mikula, Karthik Malyala]**

**Park Sense**

**Software Requirements Specification Document**

**Version: 1.0** **Date: (04/26/2022)**

**Table of Contents**

1. Introduction

1.1 Purpose

1.2 Scope

1.3 Definitions, Acronyms, and Abbreviations

1.4 References

1.5 Overview

2. The Overall Description

2.1 Product Perspective

2.1.1 System Interfaces

2.1.2 Interfaces

2.1.3 Hardware Interfaces

2.1.4 Software Interfaces

2.1.5 Communications Interfaces

2.1.6 Memory Constraints

2.1.7 Operations

2.1.8 Site Adaptation Requirements

2.2 Product Functions

2.3 User Characteristics

2.4 Constraints

2.5 Assumptions and Dependencies

2.6 Apportioning of Requirements

3. Specific Requirements

3.1 External interfaces

3.2 Functions

3.3 Performance Requirements

3.4 Logical Database Requirements

3.5 Design Constraints

3.5.1 Standards Compliance

3.6 Software System Attributes

3.6.1 Reliability

3.6.2 Availability

3.6.3 Security

3.6.4 Maintainability

3.6.5 Portability

3.7 Organizing the Specific Requirements

3.7.1 System Mode

3.7.2 User Class

3.7.3 Objects

3.7.4 Feature

3.7.5 Stimulus

3.7.6 Response

3.7.7 Functional Hierarchy

3.8 Additional Comments

4. Change Management Process

5. Document Approvals

6. Supporting Information

# 1. Introduction

## 1.1 Purpose

The purpose of this Software Requirements Specification document is multifold:

1. To specify the requirements that need to be worked on for the project to be successful.
2. To come up with structure for how the software process will be for the project.

This document is intended for the developers to have a cohesive collection of what to expect and for others to be able to view and get a high-level understanding of the project.

## 1.2 Scope

The software product being produced is called *Park Sense*. *Park Sense*will use video footage, along with edge detection for object identification, to keep track of when parking spots are available in a parking lot. The data collected will be used to create useful information for users about parking conditions which they will be able to access in a web app. Some additional goals that we would like to achieve, time permitting, is to have some data analysis to potentially track historical data to come up with statistics like when spots are most likely open and where they are open more often. We would also like to try and transition to having both a web app and a mobile app version that users can decide between.

## 1.3 Definitions, Acronyms, and Abbreviations.

* MongoDB: a NoSQL database program that uses JSON-like documents
* Node.JS: a backend JavaScript runtime environment that executes JavaScript code outside of a web browser.
* Raspbian OS: the operating system for a Raspberry Pi device.
* OpenCV: a programming library mainly aimed at real-time computer vision.
* Python3: the third major version of Python.
* AWS: Amazon Web Services

## 1.4 References

* Canny Edge Detection: <https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html>
* OpenCV-Specific Edge Detection: <https://learnopencv.com/edge-detection-using-opencv/>

## 1.5 Overview

The rest of this document is organized into two other sections, each of which may hold more value than the other depending on the information one may be concerned with. The following sections are the following:

* Section 2: General product description, user interface ideas, dependency analysis, what is needed for set up, etc.
* Section 3: Technical requirements needed for development of the product

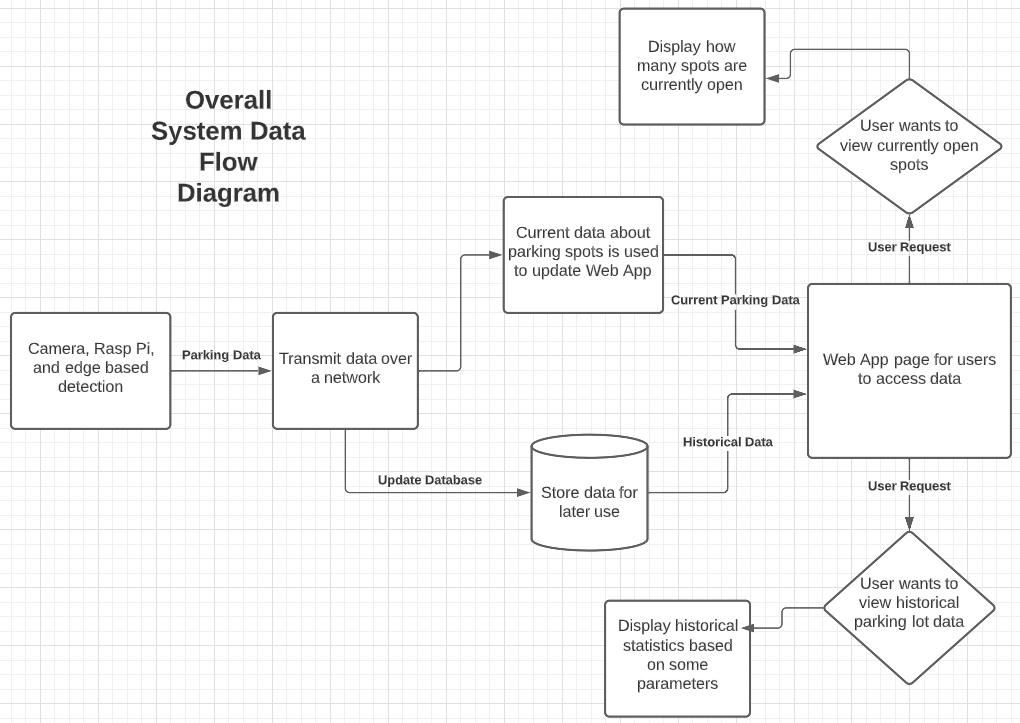
# 2. The Overall Description

This *Park Sense* software is designed to be a useful tool for places with typically limited parking spaces like colleges or some apartment complexes. The end goal is to be a useful tool that will help users be able to view available parking spaces and view historical data of parking spaces to reduce time wasted driving around to find a parking spot that is available. This should, in turn, reduce traffic congestion around campus if adoption is widespread.

This software solution was thought of as the problem that it seeks to solve has been a frequent complaint from many of the people working on the project as well as many of their peers and acquaintances over the years. Upon completion of the software, we hope to reduce these frustrations for anyone who uses the *Park Sense* product.

## 2.1 Product Perspective

High Level System Overview:



The *Park Sense* software is a mostly self-contained system, but edge detection with a Gaussian Curve and the Canny Algorithm will be used to help with the identification of vehicles and empty parking spots. We will also interface with a MongoDB database to store data needed for users. The Python Library that will be utilized to process the computer vision is OpenCV. It allows for applying a Gaussian blur to the images received from the camera/webcam and further allows for image processing and detection. This library and the Python scripts will be implemented with a Raspberry Pi for portability and efficiency to handle the data and later transfer as needed. The project was completed with approval from Dr. Harry Zhang as part of the CSE 550: Software Engineering course at the University of Louisville’s J.B. Speed School of Engineering. Users will have the ability to access *Park Sense* in a Web App and potentially on a Mobile App if time permits. This means that it would be possible for users to not need any extra software if using the Web App version of the software as they would only need a web browser. The software system will also need to integrate with a Raspberry Pi and a camera to capture the needed footage of parking lots and send the data over a network.

### 2.1.1 System Interfaces

As specified in Section 2.1 the software portion of *Park Sense* is a mostly self-contained system, only using a pre trained network to help identification of vehicles and a MongoDB database to store the necessary data. The overall system will require some interfacing with the devices used for obtaining video footage, a Raspberry Pi, and a server. The server will be hosted on Amazon Web Services. The system server will include the following:

* A MongoDB database connection to store data that will later be used to make estimations of when parking spots are usually open based on trends .

### 2.1.2 Interfaces

End users will have one point of interface with the software: an application accessible via a web browser or on mobile devices (smartphones, tablets, etc.) that will allow the user to look at information about different parking lots and inform them of where spots are currently vacant. The interface that users will interact with will be a web or app-based GUI which will only have a few options to select from. The GUI will display useful and relevant information to the users concerning the parking lot they want to view in an easy-to-understand format.

The application UI will need to update frequently and quickly enough for users to know which parking spots are available based on recent information. The UI in the application will need to be intuitive and polished to the extent that users will know which parking spots are available at a quick glance.

### 2.1.3 Hardware Interfaces

This project will utilize a Raspberry Pi 3 Model B as its primary microcomputer for image processing with an external webcam or camera. The Pi will need to access and interact with the external camera using a simple USB Cable medium which will then be passed as a parameter into the OpenCV Image instance that we create in the Python Script. The Pi will need a simple micro-USB power source and the external webcam connected via USB cable in terms of physical connections. The team can then SSH into the Pi and run the scripts accordingly after mounting the system in a specified location for best viewing angles of the parking lot.

### 2.1.4 Software Interfaces

There is no customer specified Software Interfaces required.

### 2.1.5 Communications Interfaces

No custom interfaces are being used for this software system.

HTTP will be utilized for network communication; this is a common and standard protocol.

### 2.1.6 Memory Constraints

The Raspberry Pi 3 Model B comes with 1 GB RAM and a Cortex-A53 ARMv8 64-but SoC @ 1.4 GHz architecture. The above specifications will allow for basic edge detection but not anything too advanced like using a Convolutional Neural Network (CNN). This will be the only memory constraint in terms of hardware. In terms of software,there are no memory constraints as more hard-drive and RAM can be added to the server as need be.

### 2.1.7 Operations

The user will only be required to select if they want to view the current parking lot information or the historical data for a parking lot to see statistics of when spots are usually open. The goal is to make the user experience as intuitive as possible so that they will not need any specific knowledge of the software to effectively use it. Data processing will all be done by the software and the necessary information will be available for users of the software to view and use as they need. The database will be stored on a machine for the purposes of the project but if this were to be scaled up it would likely be held on a server with backups of the historical data occasionally being taken by the server owner. Updates to the software or movement of data to a different server would likely be scheduled as needed.

### 2.1.8 Site Adaptation Requirements

To upscale the *Park Sense* system to parking lots other than the ones used in initial testing there would need to be a few steps taken per parking lot. The following are the steps that need to be taken:

* More cameras and Raspberry Pi’s would need to be installed in useful locations to be able to have a view of the parking lot to be added.
* The Raspberry Pi would need to be connected over a network so data could be sent and received.
* A new database table would need to be created for every parking lot so information between parking lots would not be overlapping when not needed.
* Setup for parking spaces would need to be done so that the edge detection technique would know where the parking spaces are
* The camera and Raspberry Pi would need to be linked to the correct database table so that the data gets sent to the correct one.

## 2.2 Product Functions

The software product being produced is called *Park Sense*. *Park Sense*will use video footage, along with deep learning for object identification, to keep track of when parking spots are available in a parking lot. The data collected will be used to create useful information for users about parking conditions which they will be able to access in a web app. Some additional goals that we would like to achieve, time permitting, is to have some data analysis to potentially track historical data to come up with statistics like when spots are most likely open and where they are open more often. We would also like to try and transition to having both a web app and a mobile app version that users can decide between.

**2.2.1) High-level requirements**

**2.2.1.1) Hardware will gather data to be used by OpenCV for image processing [Version 1.0]**

The video recorder device and Raspberry Pi will work together to obtain video data which will be analyzed by OpenCV to enable edge detection.

**2.2.1.2) Edge detection will be used to identify vehicles and empty parking spaces [Version 1.1]**

An edge detection algorithm, Canny Algorithm to be specific, will be implemented to identify parking spaces that are occupied by vehicles and empty parking spaces. The Python script will output the results that it reads which would be the number of occupied and free spots and this output will be sent over a network to be used.

**2.2.1.3) Send output neural network data over a network to be stored on a server [Version 1.0]**

Once the edge detection can effectively identify vehicles and empty parking spaces and output how many vehicles there currently are, the Raspberry Pi will need to send the output data over a network to be stored on a server. This is so that the data can be processed and put into a useful form for users to access.

**2.2.1.4) A database will be created to store relevant data [Version 1.1]**

A database will need to be created to store relevant information that is output from the identified spots after edge detection.

**2.2.1.5) Server data will be put into a database for storage and the ability to perform queries [Version 1.1]**

Once the data is transferred to a server it will be put into a database so that it can be stored in an easy-to-understand way and queries can be performed on the data.

**2.2.1.6) User will be able to access *Park Sense* with a web app [Version 1.0]**

*Park Sense* will be available through a web application which users will be able to access through a web browser.

**2.2.1.7) User will be able to access data that shows how many spaces are currently available in a parking lot [Version 1.0]**

There will be a page in the web application where users can view how many available spots there currently are (in between refresh times) in a parking lot.

**2.2.1.8) Users will have access to data that shows historical data of a parking lot [Version 1.0]**

There will be a page in the web application where users can view some historical data of a parking lot. An example would be that a user selects a time frame and they could see how many spots are typically open in a parking lot at that time.

## 2.3 User Characteristics

There will be one main class of users for the initial run of the software system but there is the possibility of another user class being added with time permitting. The first, essential, user class will be unauthorized everyday users of the software. The second, optional, class of users will be system administrators.

Almost all users will be classified as unauthorized, and this will be the default for people who quickly browse the Web App or Mobile App. Unauthorized users will be able to select if they want to view the current status of a parking lot or the historic statistic data of a parking lot. The goal is also to make it so a user can select which parking lot they would like to view if there are multiple parking lots to select between for one site, this will be an optional goal though since testing and time will be limited for the first run of this project. An unauthorized user will only need minimal technical expertise, as they will only be responsible for being able to use a web browser to access the Web App version or install a Mobile App to use the mobile version. There should not be any extra technical expertise needed to use *Park Sense* as there will only be a few options for unauthorized users to select from to see the data that they are interested in.

The second class of users will be system administrators. As mentioned earlier, this group will only be implemented if time allows as it is not as vital to get the proof of concept down but would be extremely helpful to convert *Park Sense* into a more scalable, commercial product. A system administrator would have access to set up new parking lots to their system and get the data flowing in and set up. A system administrator would need some technical expertise as they would be responsible for setting up the video recorders, the Raspberry Pi’s, getting them connected to the network, marking the parking spaces for edge detection, and setting up a database table for each new parking lot. Some help would be provided as they would be able to use a similar, but modified, database script to create new tables for parking lots and the process to set up new devices and network connections would be like all others. They will also need the basic knowledge required of unauthorized users to use a web browser or app to make sure that the changes to the system are successful.

## 2.4 Constraints

* Data will be stored in a relational database so that historical data can be used for statistics and viewed by users.
* The initial range of parking lots will be limited due to limits on where cameras can be positioned without special permission.
* The video data will only be periodically sent to reduce the amount of data being processed due to being on a Raspberry Pi and not wanting to fill up storage space on the computers being used quickly.
* The total available parking spaces must be pre-defined to then apply edge detection. This means that the implemented technology cannot be placed on any other lot and expect it to work accurately without setup.

## 2.5 Assumptions and Dependencies

With regards to hardware, the team is dependent on the single-board computer acting static throughout its operation for easy debugging. In order to access the board remotely through SSH, the board must have a static IP Address that is connected to a strong network for data communication. However, when a network resets, there is a high chance the IP Address for the board changes as well, unless specified otherwise. When the IP Address changes, we would then need to unmount the system and retrieve the new IP address again. The software assumes a Linux OS running in a cloud hosting environment that will always provide access with no down time.

## 2.6 Apportioning of Requirements.

**2.6.1) Have multiple cameras and Raspberry Pi systems connected over one network in universities or complexes with multiple or large parking lots [Version 1.0]**

In many cases, one video recorder and Raspberry Pi will not offer enough coverage for all available parking. A system will be created where multiple devices can be on the same network and contribute to the database for an area.

**2.6.2) In the *Park Sense* web application a user will be able to select the specific parking lot or section of a large parking lot [Version 1.0]**

A user will be able to select which parking lot or area of a parking lot they want information on. This will be helpful if there are multiple parking lots like for a university or if there is a very large parking lot that one video recorder could not reliably capture.

# 3. Specific Requirements

## 

## 3.1 External Interfaces

Nothing to add here at this time.

## 3.2 Functions

**3.2.1)**

The system shall output data and convert it to a 2D array data structure

The system shall take that data structure and feed it into the MongoDB database

**3.2.2)**

The AWS server shall run a MongoDB server instance

The MongoDB server shall make new database entries when requests are made

The MongoDB server shall allow editing of database entries

The MongoDB server shall store historical data about the parking spots

**Web Applications Requirements**

**3.2.3) Web Application will pull all needed data from a database [Version**

**1.0]**

The Web Application for *Park Sense* will need to interface with a database for

historic statistical data. The data accessed from the database will be used for the

following:

* Viewing the average number of available parking spots during certain times of the day
* Viewing the average number of available parking spots on specific days (like on Mondays or on Saturdays)

This data will need to be queried for and accessed when a user wants to view the historical data for a parking lot. The process for accessing the data should follow a process like the following:

1. User clicks on “View historical data” options
2. User is presented with what kind of data they would like to view
   1. Select a time range
   2. Select a day of the week
   3. Select a combination of the above two
3. Depending on the user's selections the database will then be queried in such a way that the data requested will be available
4. After the DB is queried, the information will be displayed to the user in an easy-to-understand format.

**3.2.4) Web Application will display the total number of parking spaces in a parking lot and how many parking spots are currently available. [Version 1.1]**

For each parking lot monitored by the *Park Sense* system the maximum number of parking spaces will be made known to the users and the number of available parking spaces will also be available to the users.

* The available parking spaces will be updated at a defined rate (fifteen seconds is the current test standard) from output data sent from the edge detection method.
* The ratio of open parking spaces to total parking spaces will be displayed to the user in a simple to understand format. A simple example of how this could be displayed would be like: “There are currently 4 parking spaces available”. “4/10 parking spaces are currently available”

## 3.3 Performance Requirements

**3.3.1) *Park Sense* will be able to handle at least 10 simultaneous users on the**

**Web Application at a time [Version 1.0]**

Starting out we would like the *Park Sense* Web Application to be able to handle

at least 10 users simultaneously This is just to get things started and will

hopefully be able to be scaled as needed.

**3.3.2) The Raspberry Pi will need to send data on the**

**observed parking lot(s) every fifteen seconds. [Version 1.1]**

Data will need to be reliably sent every fifteen seconds maximum so that there

will not be any very large gaps in between when data is gathered. This will make

the system more appealing and accurate for the users.

* The edge detection algorithm will determine how many vehicles are currently in the parking lot and how many available spaces there are.
* Once the image classification outputs its data, it will be sent over a network to a server/device for storage.
* The data will be stored in a database for future use.

## 3.4 Logical Database Requirements

**3.4.1) Data will be received over the network from the edge detection outputs and translated into useful data to be stored into the database [Version 1.0]**

The edge detection algorithm will output data of how many vehicles are currently occupying a parking lot. This data will be broken down into the following information:

* The number of parking spaces that are being used at the time of data transmission
* The number of currently available parking spaces in the parking lot
* The date, day of the week, and time of when the data is sent to be used in future historical data statistics

**3.4.2) Database will have data added to it every fifteen seconds [Version 1.1]**

Output edge detection data will be sent from the Raspberry Pi to be stored in the

database every fifteen seconds. This is so the amount of data that needs to be

stored can be kept reasonable while also refreshing at a reasonable rate so that the

data can be accurate in small time frames.

**3.4.3) Database will need to be accessed when providing historic statistical**

**Data [Version 1.0]**

When a user wants to view the historical statistics of a parking lot the database

will need to be queried to return the relevant data.

**3.4.4) Data in the database will be deleted after a specified amount of time**

**[Version 1.0]**

To prevent the database from growing too large, the data that is stored will be

deleted after an allotted amount of time has passed since the data was first stored

in the database. This could be something that is configurable or a static amount of

time like “Delete data that is over 6-months old”. The point is to keep enough

data so that it is useful but to not keep so much that storage becomes an issue or

out of date.

## 3.5 Design Constraints

### 3.5.1 Standards Compliance

No preexisting standards compliance is needed for the *Park Sense* software system.

## 3.6 Software System Attributes/Non-Functional Requirements

**Web App UI Requirements**

**3.6.1.1) The *Park Sense* Web App will display capacity statistics graphically**

The main focus of the UI will be to display capacity statistics such as: total spots available, current capacity, and number of spots available at a certain time based upon historical data. These statistics will be displayed graphically and will include the capability to switch which statistics they are viewing through buttons. The graphs will be updated in real time and will include numerical labels for clarity.

**3.6.1.2) The *Park Sense* Web App will include a search capability for users to select a specific parking lot on campus.**

The UI will also include a functional search bar to demonstrate a possible capability for the future of searching and selecting a specific parking lot to look at. Users will be allowed to select a parking lot by name and when selected the app will display the corresponding availability statistics.

### 3.6.2 Reliability

*Park Sense* software will require the Raspberry Pi to maintain constant internet connection with speeds sufficient to upload live video footage and/or intermittent frames and data regarding parking availability detection. Web service servers where information is contained regarding parking lot data and host of web application will need to remain online. User devices will need internet connection to fetch updated parking information on the application.

### 3.6.3 Availability

**3.6.3.1) The *Park Sense* System will run when the Raspberry Pi, Video**

**recorder, database, and server are available.**

Due to the nature of the system for the class, *Park Sense* will only be available

while all of the required hardware and the database is online. If this were to be

upscaled the availability could be increased by having a central server and an

embedded device that does not need to be plugged in or have batteries changed

frequently like Raspberry Pi and the video recorder.

### 3.6.4 Security

Data modifiers will be inaccessible by end users and historical data will be kept for reference of 1) user knowledge of parking lot availability and 2) developer knowledge of data accuracy and security.

Other than this, there are no outstanding security issues to address.

### 3.6.5 Maintainability

Outside of good practices, the developers are unsure of what requirements would fit into this category as they are uncertain of whether this software will be developed beyond the prototype. So, maintainability may not be of question.

### 3.6.6 Portability

*Park Sense’*s web application will be developed using mobile-friendly languages

such as Python and JavaScript. The web application will be accessible from iOS

and Android mobile operating systems.

# Change Management Process

For any modifications to the SRS the team will need to reach a consensus on if a section or requirement needs to be changed. If it is agreed that a change is needed then the date and original information will be recorded for future reference.

# Document Approvals

This section does not apply to this software system as it is for a college course.

# Supporting Information

Nothing to add here at this time.