University of Waterloo
Faculty of Engineering
Department of Electrical and Computer Engineering

A Distributed Garbage Collection System

ECE 498B Design Project Final Report

Course: ECE 498B

Group ID: 2015.034

Group Member Name	**UW Id/user#**
Rahul Sharma	r54sharm/20382012
Justin Cheung	j43cheun/20390063
Ling Xing	ly2xing/20378372
Zored Ahmer	zbahmer/20379543
Evgeny Nam	e5nam/20375789

Project Consultant: Jeff Zarnett

Date of Submission: Friday, February 13, 2015

Acknowledgements

We would like to thank Dr. Alaa Khamis for providing technical advice regarding the design and implementation of the garbage bin allocation algorithm. Additionally, we would also like to thank Sanjay Singh for designing the case to house the hardware components of our project. Finally, we would like to thank Jeff Zarnett for advising our capstone project and following up on us to ensure that we are on the right track. We acknowledge that we have received no further help in the development of the Distributed Garbage Collection System other than those mentioned above in writing.

Table of Contents

List of Figures

List of Tables

Abstract

Conventional garbage bins are static receptacles for temporarily storing non-reusable and non-recyclable waste. Today, these bins are stationed at arbitrary locations and serviced by waste collectors on a fixed schedule and route. Given these limitations, time is allocated poorly such that receptacles in areas of low usage are serviced too frequently and receptacles in areas of high usage are serviced too infrequently, leading to overflow and more litter over a shorter time span. To overcome these issues, the Distributed Garbage Collection System aims to provide an active approach towards garbage collection. Effectively, this system is a network of garbage bins, where each bin is stationed on a patrol cluster and capable of asynchronous load detection and location identification via GPS. These functions are implemented through a combination of sensors and embedded software design. Additionally, wireless communications and data storage using knowledge from computer networks and databases enables each garbage bin to transmit load data and location coordinates to a central server for performing garbage bin allocation. Compared to existing solutions, this system automatically schedules the collection of garbage when full and strategically allocations bins based on usage statistics, which should reduce littering and garbage collection costs in the long run.

1. High-Level Description of Project

1.1 Motivation

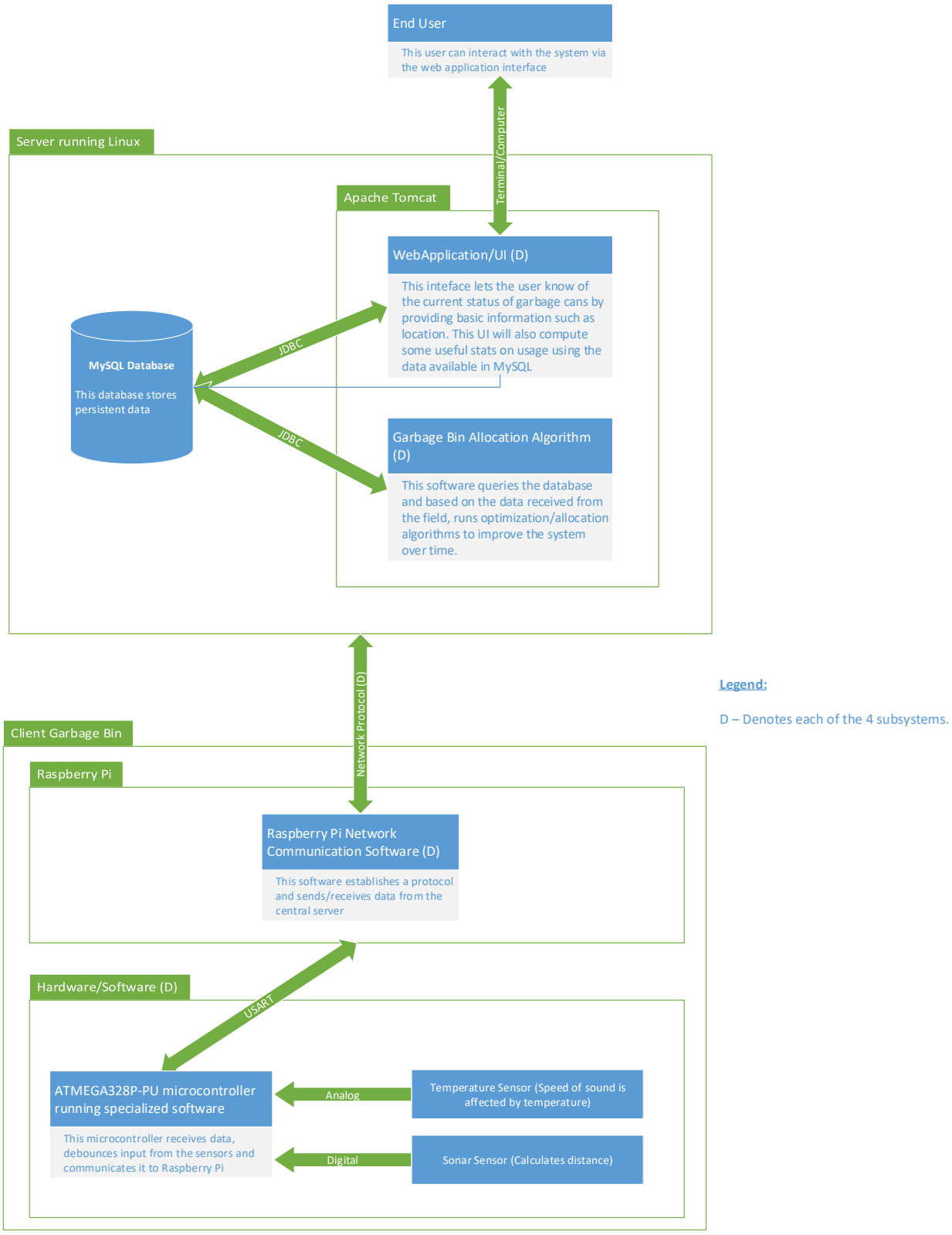
The allocation of garbage receptacles in outdoor settings at times appear to be inefficient or arbitrary, with many bins having been placed on the assumption of where they are needed most. Additionally, servicing these receptacles requires maintenance and park staff be on a common reoccurring schedule and route, often regardless of whether or not these bins are filled to capacity. Because of this common scheduling, it can often be seen when attending a closed outdoor venue that garbage bins are often either left overflowing for some time before they are serviced or are emptied prematurely. This in turn means either loss of resources or productivity, or results in the act of littering by park guests and campus visitors. In fact, a number of official reasons can be attributed specifically to the existence and use of garbage bins [1] [2]. Whether bins are in the wrong place at the wrong time, not emptied out often enough, or there are simply not enough bins at that instance, the Distributed Garbage Collection System offers added convenience to users while actively providing a means to prevent the generation of litter so that individuals may never concern themselves with what to do with their trash.

1.2 Project Objective

Given the problem of litter and poorly allocated garbage collection resources, the Distributed Garbage Collection System sets out to provide a solution by means of hardware and software. Specifically, the objective holds that by providing a number of garbage receptacles communicating in a network, it can result in reduced garbage and litter, and aid in an efficient resource allocation of garbage collection operations. These garbage bins will ultimately be able to detect capacity by volume. At this point, these bins will be able to notify a central server of their complete capacity. The central server will then be able to take the appropriate action to fill the vacant patrol cluster by reallocating available garbage bins based on usage statistics in specific locations.

These bins essentially allow for the elimination of inefficiencies found in human controlled garbage collection protocols in closed outdoor settings, such as theme parks and university campuses. Deployment is determined using knowledge of past usage statistics and replacement is scheduled automatically by solving two of the obvious problems (the where and when) that come to mind when one thinks of garbage collection. Although alternatives may exist for the problem of garbage collection (perhaps routed garbage chutes), the Distributed Garbage Collection System offers a simpler, more cost effective answer using electrical and computer engineering design concepts from areas such as cooperative and adaptive algorithms, distributed computing, computer networks, and statistics.

1.3 Block Diagram



Legend:

D – Denotes each of the 4 subsystems.

2. Project Specifications

2.1 Functional Specifications

Functional Specification		Essential (Yes/No)
1	There must be a latency of under 5 seconds when measuring garbage volume.	Yes
2	The garbage bin must send to the server every 5 minutes data pertaining to the current volume of garbage contained inside it and its current location.	Yes
3	The garbage bin sensors subsystem must be able to measure and report depth measurements that are within ±10% accuracy of a manually produced measurement.	Yes
4	The user interface on the server should display the status of each garbage bin currently deployed into the field.	Yes
5	If a garbage bin is within 90% to 100% filled to capacity, the user interface should indicate that the garbage bin is full.	Yes
6	If a garbage bin is considered to be full, the server should exclude it from the next garbage bin allocation.	Yes
7	The cost to be minimized by the garbage bin location search algorithm must account for the distance from the garbage bin location to each garbage spot in the corresponding cluster of garbage spots.	Yes
8	The cost to be minimized by the garbage bin allocation algorithm must account for the distance from each garbage bin to its assigned garbage cluster, the distance from each cluster to the nearest service station, and the remaining volume in each garbage bin.	Yes
9	The garbage bin allocation subsystem must be able to perform garbage bin allocation on a non-homogeneous set of garbage bins and garbage spots.	Yes
10	The garbage bin location search and garbage bin allocation algorithms are best-effort and do not guarantee convergence to the lowest cost solution.	Yes
11	Together, the garbage bin location search and garbage bin allocation algorithms must be scalable to at least 1000 garbage spots and 100 garbage bins.	Yes
12	For any given garbage bin allocation round, the number of garbage bin locations generated by garbage bin location search must not exceed the number of available bins.	Yes
13	Together, the runtime of the garbage bin location search and garbage bin allocation algorithms must not exceed 5 minutes.	Yes

2.2 Non-Functional Specifications

Function Specification		Essential (Yes/No)
1	The sensors and associated circuitry must not obstruct normal operation. The incoming trash must not touch the sensor. There must be a clearance of at least 1cm.	Yes
2	The system should be scalable and support at least 70 garbage bins. This can be verified via simulation.	Yes
3	The garbage bin sensors hardware should fit in a 20cm x 20cm x 15cm enclosure.	No

4	A user should not need to spend more than 10 seconds to verify the status of a bin.	No

3. Detailed Design

3.1 Garbage Bin Allocation Subsystem

3.1.1 Problem Characterization

The first subsystem developed in this project is a garbage bin allocation algorithm for allocating garbage spots (i.e., locations where people have littered in the past) to garbage bins in an open space. To develop this subsystem, the first design iteration involves characterizing the optimization problem to be solved relative to the following use case. In this use case, it is assumed that the system has been running for a long time. Within the current time instance, several garbage bins currently deployed into the field have indicated that they need to be serviced. Hence, the server must now determine how it is going to allocate these vacated garbage spots to the remaining available garbage bins. To complicate matters, the set of available garbage bins and the set of garbage spots are not necessarily homogeneous. In particular, each available garbage bin can be differentiated by its location and its remaining garbage volume. Likewise, each garbage spot can be differentiated by its location and its average volume of production of garbage within the current hour. Furthermore, the typical use case suggests that there will usually be many more garbage spots than garbage bins.

Since the quantity of garbage spots will typically exceed the quantity of available garbage bins, a 1:1 allocation of available garbage bins to garbage spots is not possible. As a result, the first requirement is determining the geographical locations where garbage bins should be placed on the map. In this case, the quantity of these garbage bin locations must not exceed the number of available garbage bins. Furthermore, each of these garbage bin locations should ideally be positioned in such a way that it is evenly distanced from the subset of garbage spots closest to it (i.e., it is the closest garbage bin location to these garbage spots), while at the same time, minimizing the total aggregate distance of each garbage spot to the garbage bin location closest to it over all garbage spots.

After determining these garbage bin locations, the next task is to take care of allocating each of the available garbage bins to each of these garbage bin locations. Since the set of available garbage bins and the set of garbage spots are potentially heterogeneous, certain allocations of garbage bins to garbage bin locations may be more expensive than others. Hence, the garbage bin allocation algorithm must somehow be able to quantify the cost associated with any particular allocation of garbage bins to garbage bin locations so as to minimize the overhead of garbage collection. Such a multi-objective cost function should account for the distance of each garbage bin to their designated garbage bin location and the distance from the garbage bin location to the closest servicing station such that higher costs are applied for greater distances. Additionally, this multi-objective cost function should impose a cost penalty that increases with the likelihood that a garbage bin location produces more garbage in the current hour than can be held by its allocated garbage bin.

3.1.2 Problem Formulation

Based on the problem characterization provided in Section 3.1.1, the garbage bin allocation problem can be formulated as the following two optimization sub-problems:

- Let $B = \{b_1, \dots, b_k\}$ represent the set of available garbage bins, where $1 \leq k = |B|$.
- Let $G = \{g_1, \dots, g_l\}$ represent the set of garbage spots, where $1 \leq l = |G|$.
- Let $P = \{p_1, \dots, p_m\}$ represent the set of garbage bin locations, where $1 \leq m = |P| \leq k = |B|$.
- Let $C = \{C_1, \dots, C_m\}$ represent the set of garbage clusters, where $1 \leq m = |C| = |P| \leq k = |B|$, each garbage cluster $C_i \in C$ is a disjoint subset of G (i.e., for $C_i, C_j \in C$, where $i \neq j$, $C_i \cap C_j = \emptyset$), and the closest garbage location to each garbage spot $g \in C_i$ is the garbage bin location p_i.
- Let $Q = \{q_1, \dots, q_n\}$ represent the set of service station locations, where $1 \leq n = |Q|$.
- Let h_i represent the service station nearest geographically to the geographic centroid of garbage cluster C_k for $1 \leq i \leq k$, where $h_i \in Q$.
- Let v_i represent the remaining volume in garbage bin g_i for $1 \leq i \leq k$.
- Let λ_i represent the aggregate sum of the average volume of garbage produced by each garbage spot g in garbage cluster C_i over all garbage clusters $C_i \in C$.

Given the parameters defined above, the objective of the first optimization problem is to find a set of garbage bin locations,P, that minimizes the following cost function:

$$clusterCost(P) = \sum_{i=1}^{m} \sum_{g \in C_i} (distance(g, p_i))^2 \tag{1}$$

Meanwhile, the objective of the second optimization problem is is to find an allocation $A = \{(b_1, p_1), \dots, (b_m, p_m)\}$ that minimizes the following cost function:

$$allocationCost(A) = \sqrt{(totalDistance(A))^2 + \big(overloadPenalty(A)\big)^2} \tag{2}$$

where

$$totalDistance(A) = \sum_{i=1}^{k} distance(b_i, p_i) + distance(p_i, h_i) \tag{3}$$

$$overloadPenalty(A) = 100 \cdot \sum_{i=1}^{k} \Pr(X > v_i) = 100 \cdot \sum_{i=1}^{k} 1 - \Pr(\lambda_i \leq v_i) \tag{4}$$

$$\Pr(X \leq v_i) = e^{-\lambda_i} \cdot \sum_{j=1}^{\lfloor v_i \rfloor} \frac{\lambda_i^j}{j!} \tag{5}$$

In terms of formulas, formula (1) defines the cost function to be minimized by the garbage bin location search algorithm. Meanwhile, formula (2) defines the cost function to be minimized by the garbage bin allocation search algorithm. In the case of formula (1), the cost of a given garbage bin location set, P, is defined as the aggregate sum of the squared distance from each garbage spot to the closest garbage bin location over all garbage spots such that higher costs are yielded for greater distances. In the case of formula (2), the cost consists of two components: the total distance (computed using formula (3)) and the overload penalty (computed using formula (4)). The total distance component represents the total aggregate distance in kilometers that must be travelled in order to move each available garbage bin from its current position to the designated garbage bin location and from the designated garbage bin location to the nearest service station. Meanwhile, the overload penalty represents an additional cost that is meant to penalize a given allocation for each garbage bin that has a high probability of overfilling. In particular, each garbage bin b_i has a remaining garbage volume v_i that preferably should not be exceeded. The probability that v_i is exceeded, as accounted for in formula (4), is modelled using the cumulative Poisson distribution defined by formula (5). Given formulas (4) and (5), allocations where garbage bins have a higher probability of overfilling will incur a higher overload penalty.

It is important to note that all positions represented in the problem formulation above are intended to be represented as a $[latitude, longitude]$ pair, where $-90 \leq latitude \leq 90$ is the latitude in degrees and $-180 \leq longitude \leq 180$ is the longitude in degrees. Given this representation for location, the distance components shown in formulas (1) and (2) (as denoted by $distance()$) are intended to be computed using the Haversine formula, as documented in [3]. For the sake of completeness and clarity, the algorithmic representation of the Haversine formula that is used to compute the distance components in formula (2) is reproduced below:

Algorithm 1: Pseudocode for Computing Distance in Kilometers Using the Haversine Formula [3]

```
Input Parameters:
1   startLoc:           The start location.
2   destLoc:            The destination location.

Output Parameters:
3   distanceKm:         The distance between startLoc and destLoc in kilometers.

Constant Definitions:
4   earthRadiusKm:      The radius of the Earth in km.

Variable Definitions:
5   startLatRads:       The latitude of startLoc in radians.
6   startLongRads:      The longitude of startLoc in radians.
7   destLatRads:        The latitude of destLoc in radians.
8   destLongRads:       The longitude of destLoc in radians.
9   ΔlatRads:           The difference between destLatRads and startLatRads.
10  ΔlongRads:          The difference between destLongRads and startLongRads.
11  a                   An intermediate variable for the Haversine formula.
12  c                   An intermediate variable for the Haversine formula.

Algorithm:
13  earthRadiusKm ← 6371
14  startLatRads ← startLoc.latitude() × π / 180
15  startLongRads ← startLoc.longitude() × π / 180
16  destLatRads ← destLoc.latitude() × π / 180
```

```
destLongRads ← destLoc.longitude() × π / 180
ΔlatRads ← destLatRads – startLatRads
ΔlongRads ← destLongRads – startLongRads
a ← sin²( ΔlatRads / 2 ) + cos( startLatRads ) × cos( destLatRads ) × sin²( ΔlongRads )
c ← 2 × atan2( sqrt( a ), sqrt( 1 – a ) )
distanceKm ← earthRadiusKm × c
```

3.1.3 Problem Modelling and Design Alternatives

Now that the optimization problem has been characterized and formulated, the next step in terms of design iteration is to find an appropriate model for solving it. In particular, one of the concepts learned in the fourth year ECE course, ECE 457A Cooperative and Adaptive Algorithms, is that an optimization problem in practice can be modelled after other problems. Since the garbage bin allocation problem is formulated as two separate optimization sub-problems in Section 3.1.2, two problem models will need to be defined in order to solve the garbage bin allocation problem. From these two problem models, the actual design in this subsystem involves developing solution models to each of these problems.

3.1.3.1 Decomposition of Garbage Bin Location Search to the K-Means Clustering Problem

As alluded to in the problem formulation in Section 3.1.2, the optimization problem for garbage bin location search closely resembles the k-means clustering problem [4]. In the k-means clustering problem, the following inputs are provided: a set X of $n > 0$ data points and a positive integer k, where $0 < k \leq n$, denoting the number of clusters to be created [4]. From these input arguments, the objective of the classic k-means clustering problem, according to [4], is to find a set C of k cluster centroids (i.e., a centroid is the center point in a cluster) that minimizes the following cost function:

$$kmeansCost(C) = \sum_{x \in X} \min_{c \in C} \|x - c\|^2 \tag{6}$$

It is important to note here that the k-means cost function defined by formula (6) only accounts for the squared distance (represented as a L2-norm) between each data point $x \in X$ and the cluster centroid $c \in C$ closest to that data point [4]. Consequently, each data point $x \in X$ is assigned to the cluster centroid $c \in C$ closest to it, thus creating a set of k clusters out of the set X of $n > 0$ data points [4].

Given the definition of the k-means clustering problem, it becomes evident that garbage bin location search is an application of k-means clustering. In particular, the set of garbage spots G in garbage bin location search closely parallels the set X of $n > 0$ data points in the k-means clustering problem [4]. Likewise, the set of garbage bin locations to be found in garbage bin location search closely parallels the set C of k cluster centroids to be found in k-means clustering [4]. Finally, the cost function for garbage bin location search (see formula (1)) is effectively the cost function for k-means clustering (see formula (6)) with the L2-norm replaced by Haversine distance [4]. Due to these parallels, the design of a solution for garbage bin location search will consider the application of two algorithms typically applied towards solving the k-means clustering problem: the k-means algorithm and the k-means++ algorithm [4].

3.1.3.2 Decomposition of Garbage Bin Allocation to the Multi-Robot Task Allocation Problem

In terms of the garbage bin allocation problem, it turns out that it resembles the multi-robot task allocation problem, as defined in [5] and [6]. To see why this is the case, consider the definition for the multi-robot task allocation problem from [5] and [6]:

- Let $R = \{R_1, \dots, R_n\}$ represent the set of available heterogeneous robots.
- Let $T = \{T_1, \dots, T_n\}$ be the set of tasks to be allocated to each available heterogeneous robot.
- Let $A : T \rightarrow R$ represent the allocation function that allocates that tasks in T to the available heterogeneous robots in R.

Then the objective is to find an optimal allocation $S = \{(R_1, T_1), \dots, (R_k, T_k)\}$ that minimizes some cost function $f(S)$.

Based on the definition of the multi-robot task allocation problem provided in [5] and [6], it is evident that there are very clear parallels between the garbage bin allocation problem and the multi-robot task allocation problem. In particular, the available garbage bins can be likened to the available heterogeneous robots [5]. Meanwhile, the garbage bin location to be assigned to an available garbage bin can be analogized to the tasks to be assigned to each robot [5]. The only difference that exists between garbage bin allocation and multi-robot task allocation is that the assignment of garbage bin locations (i.e., the tasks) to the available garbage bins (i.e., the robots) is restricted to a 1:1 mapping (this is an intentional consequence from using k-means clustering to determine garbage bin locations). That is, an available garbage bin can only be assigned a single garbage bin location. Other than this minor difference, the solution models used to solve the multi-robot task allocation problem should also be applicable to garbage bin allocation. Hence, the design component for garbage bin allocation focuses on choosing which solution model from multi-robot task allocation should be adopted.

3.1.3.3 K-means-based Garbage Bin Location Search

The first algorithm to be considered for solving garbage bin location search is a modified version of the standard k-means algorithm for solving k-means clustering [4]. Given the similarities between k-means clustering and garbage bin location search (see Section 3.1.3.1), the objective of this subsection will be to develop an algorithm for garbage bin location search based on the k-means algorithm; however, before this objective can be met, the k-means algorithm must first be understood.

At a conceptual level, the k-means algorithm can be defined in terms of two phases: an initialization phase and an optimization phase. In the initialization phase, the k-means algorithm fulfills the task of computing seed values for the k cluster centroids [4]. While the choice of these seed values is arbitrary, what typically happens in practice is that the seed values are uniformly drawn at random from the data set X [4]. In the optimization phase, the k-means algorithm fulfills the task of repeatedly iterating over a two-step algorithm until the set of cluster centroids, C, converges [4]. The first step in this algorithm (i.e., the assignment step) is to assign each data point $x \in X$ to the cluster corresponding to the nearest cluster centroid [4]. The second step in this algorithm (i.e., the update step) is to update each cluster centroid to the actual center point of the corresponding cluster of data points [4].

As discussed in Section 3.1.1.1, the garbage spots can be analogized as data points and the garbage bin locations can be analogized as cluster centroids; however, due to some minor differences between the k-means clustering problem and garbage bin location search and some undesirable properties inherent in the k-means algorithm, certain modifications will need to be made to the k-means algorithm in order to apply it to garbage bin location search. In particular, since the k-means algorithm has an NP-hard time complexity, an upper bound will need to be set on the number of iterations that it can run for before terminating [4]. Additionally, the L2-norm that the standard k-means algorithm typically associates with distance will need to be replaced by the Haversine formula, since the garbage spots are represented using latitude and longitude coordinates [4] [3]. A different formula will also be needed for computing the center point of a garbage cluster for the same reason.

Aside from these minor differences, there is almost a direct mapping between the k-means clustering problem and garbage bin location search. With this in mind, the proposed algorithm for garbage bin location search based on the k-means algorithm is provided below in pseudocode. For completeness, the algorithm for computing the center point of a garbage cluster is also provided.

Algorithm 2: Pseudocode for k-means-based Garbage Bin Location Search Initialization Phase

```
Input Parameters:
1   garbageSpots:        The set of garbage spots.

Output Parameters:
2   garbageClusters:     The set of garbage clusters.

Variable Definitions:
3   randNumGenerator:    A random number generator for generating values between 0 and 1.
4   generatedIdxSet:     A set containing already selected garbage spot indices.
5   garbageSpotIdx:      A temp variable for storing the current garbage spot index.

Algorithm:
6   for i ← 1 to garbageClusters.size() do
7     garbageSpotIdx ← floor( randNumGenerator.generateValue() × garbageSpots.size() )
8     if garbageSpotIdx ← 0 then
9       garbageSpotIdx ← 1
10    endif
11    while generatedIdxSet.contains( garbageSpotIdx ) = true do
12      garbageSpotIdx ← floor( randNumGenerator.generateValue() × garbageSpots.size() )
13    done
14    garbageClusters[i].garbageBinLocation ← garbageSpots[garbageSpotIdx].location
15    generatedIdxSet.add( garbageSpotIdx )
16  done
```

Algorithm 3: Pseudocode for k-means-based Garbage Bin Location Search Optimization Phase

```
Input Parameters:
1   garbageSpots:          The set of garbage spots.
2   maxIterations:         The maximum number of iterations to run algorithm for.

Input/Output Parameters:
3   garbageClusters:       The set of garbage clusters.

Output Parameters:
4   cost:                  The cost of the final set of clusters.
```

```
Variable Definitions:
   convergeFlag:          A flag to indicate whether or not garbage clusters have converged.
   nearestCluster:        A temp var for storing nearest garbage cluster encountered so far.
   previousCluster:       A temp var for storing the previous cluster of a garbage spot.
   distToCluster:         A temp var for storing distance from spot to nearest cluster.

Algorithm:
   for i ← 1 to maxIterations do
     cost ← 0
     convergeFlag ← true
     foreach garbageSpot in garbageSpots do
       nearestCluster ← getNearestCluster( garbageSpot, garbageCluster )
       if garbageSpot.garbageCluster = null then
         garbageSpot.garbageCluster ← nearestCluster
         nearestCluster.addGarbageSpot( garbageSpot )
         convergeFlag ← false
       else if garbageSpot.garbageCluster ≠ nearestCluster then
         previousCluster ← garbageSpot.garbageCluster
         previousCluster.removeGarbageSpot( garbageSpot )
         garbageSpot.garbageCluster ← nearestCluster
         nearestCluster.addGarbageSpot( garbageSpot )
         convergeFlag ← false
       endif
       distToCluster ← getDistance( garbageSpot, garbageCluster )
       cost ← cost + ( distToCluster × distToCluster )
     done
     if convergeFlag = true then
       break
     endif
     foreach garbageCluster in garbageClusters do
       garbageCluster.garbageBinLocation = getCenterPoint( garbageCluster.getGarbageSpots )
     done
   done
```

Algorithm 4: Pseudocode for Computing the Center Point of a Garbage Cluster [3]

```
Input Parameters:
   garbageSpots:          The subset of garbage spots contained in a given garbage cluster.

Output Parameters:
   centerLocation:        The center point location of a given garbage cluster.

Variable Definitions:
   latitudeRads:          A temp var for storing a latitude value in radians.
   longitudeRads:         A temp var for storing a longitude value in radians.
   hyp:                   A temp var used to convert Cartesian coordinates to lat/lng.
   x:                     A temp var storing the equivalent x-coordinate of a garbage spot.
   y:                     A temp var storing the equivalent y-coordinate of a garbage spot.
   z:                     A temp var storing the equivalent z-coordinate of a garbage spot.
   xTotal:                Stores the total magnitude of all x-coordinate values.
   yTotal:                Stores the total magnitude of all y-coordinate values.
   zTotal:                Stores the total magnitude of all z-coordinate values.
   xAverage:              Stores the mean magnitude of all x-coordinate values.
   yAverage:              Stores the mean magnitude of all y-coordinate values.
   zAverage:              Stores the mean magnitude of all z-coordinate values.

Algorithm
   xTotal ← 0
   yTotal ← 0
   zTotal ← 0
   for garbageSpot in garbageSpots do
```

```
19    latitudeRads ← garbageSpots.latitude() × π / 180
20    longitudeRads ← garbageSpots.longitude() × π / 180
21    x ← cos( latitudeRads ) × cos( longitudeRads )
22    y ← cos( latitudeRads ) × sin( longitudeRads )
23    z ← sin( latitudeRads )
24    xTotal ← xTotal + x
25    yTotal ← yTotal + y
26    zTotal ← zTotal + z
27  done
28  xAverage ← xTotal / garbageSpots.size()
29  yAverage ← yTotal / garbageSpots.size()
30  zAverage ← zTotal / garbageSpots.size()
31  longitudeRads ← atan2( yAverage, xAverage )
32  hyp = sqrt( ( xAverage × xAverage ) + ( yAverage × yAverage ) )
33  latitudeRads ← atan2( zAverage, hyp )
34  centerLocation.latitude ← latitudeRads × 180 / π
35  centerLocation.longitude ← longitudeRads × 180 / π
```

3.1.3.4 K-means++-based Garbage Bin Location Search

The second algorithm to be considered for solving garbage bin location search is a modified version of the k-means++ algorithm proposed in [4] for solving the k-means clustering problem. As with the k-means algorithm discussed in Section 3.1.3.3, the objective of this subsection will be to develop an algorithm for garbage bin location search based on the k-means++ algorithm.

Conceptually, the k-means++ algorithm is an enhanced version of the k-means algorithm. As with the k-means algorithm, the k-means++ algorithm can be defined in terms of an initialization phase and an optimization phase, with the optimization phase being completely identical between the two algorithms [4]. The actual difference between the k-means algorithm and the k-means++ algorithm lies in how each algorithm handles the initialization phase [4]. Recall from Section 3.1.3.3 that the objective of the initialization phase is to initialize each of the k cluster centroids with seed values to be operated on during the optimization phase [4]. In the k-means algorithm, the choice of these seed values is largely arbitrary; however, it is reasoned in [4] that this approach has a high probability of yielding suboptimal clusterings as an end result. To counter this, the k-means algorithm tries to heuristically select seed values from the set of data points X that are farther apart from each other in order to promote exploration [4]. This objective is achieved by execution of the following algorithm. To begin, the first seed is uniformly selected at random from the set of data points X [4]. Then for each subsequent cluster centroid up until the k^{th} cluster centroid, the seed is selected from X under a weighted probability distribution [4]. In particular, each data point $x \in X$ is selected to be a seed with probability proportional to $\frac{D(x)^2}{\sum_{x \in X} D(x)^2}$, where $D(x)^2$ denotes the squared distance from the data point x to the nearest seed in the set of already selected seeds [4]. Based on this metric, a data point $x \in X$ that is distanced further from each of the already selected seeds has a higher probability of being selected as the next seed [4]

Given this definition for the k-means++ algorithm, the next step will be to devise an implementation for the k-means++-based garbage bin location search algorithm. In this case, the analogies between garbage bin location search and k-means clustering from Section 3.1.3.3 still apply here. With this in mind, since the k-means++ algorithm only enhances the initialization phase in the k-means algorithm, the

optimization phase of the k-means++-based garbage bin location search algorithm will be identical to the optimization phase of the k-means-based garbage bin location search algorithm devised in Section 3.1.3.3 (see Algorithm (3)) [4]. As a result, this subsection will only consider the implementation of the initialization phase for the k-means++-based garbage bin location search algorithm.

In order to implement the initialization phase for k-means++-based garbage bin location search, a conceptual model is required in order to represent the weighted probability distribution used in the selection of the m garbage bin location seeds from the set of garbage spots G. For the purposes of this project, the weighted probability distribution was modelled after a hypothetical roulette wheel, with each piece in the roulette wheel corresponding to a potential garbage bin location seed $g_i \in G$ and the size of each piece proportional to the geographical squared distance from g_i to the nearest seed in the set of already selected garbage bin location seeds. Given this roulette wheel, the next seed is then determined by the piece that is landed on after spinning the roulette wheel. For completeness, a pseudocode implementation of the initialization phase for k-means++-based garbage bin location search utilizing this roulette wheel model is provided below:

Algorithm 5: Pseudocode for k-means++-based Garbage Bin Location Search Initialization Phase

```
Input Parameters:
1   garbageSpots:        A copy of the set of garbage spots.

Output Parameters:
2   garbageClusters:     The set of garbage clusters.

Variable Definitions:
3   randNumGenerator:    A random number generator for generating values between 0 and 1.
4   spotIdx:             Stores a randomly generated garbage spot index.
5   spotToClusterMap:    A hash map for mapping a garbage spot to a garbage cluster.
6   minDist:             Stores distance from current spot to nearest garbage bin location.
7   minDistSq:           Stores sq distance from current spot to nearest garbage bin location.
8   totalMinDistSq:      Stores the total minDistSquared over all spots in garbageSpots.
9   currSeed:            Stores the current garbage bin location seed.
10  rouletteWheel:       An array of floating point values representing the roulette wheel.
11  roulettePtr:         A value representing the position on the roulette wheel after a spin.
12  prevRoulettePtr:     Stores the roulette wheel position behind the current position.
13  imin:                The min index for binary search on rouletteWheel.
14  imid:                The middle index for binary search on rouletteWheel.
15  imax:                The max index for binary search on rouletteWheel.

Algorithm:
16  spotIdx ← floor( randNumGenerator.generateValue() × garbageSpots.size() )
17  if spotIdx = 0 then
18    spotIdx ← 1
19  endif
20  currSeed ← garbageSpots.remove( spotIdx )
21  garbageClusters[1].garbageBinLocation ← currSeed.location
22  for i ← 2 to garbageClusters.size() do
23    totalMinDistSq ← 0
24    for j ← 1 to garbageSpots.size() do
25      minDist ← getDistanceToClosestGarbageCluster( garbageSpots[j], garbageClusters )
26      minDistSq ← minDist × minDist
27      totalMinDistSq ← totalMinDistSq + minDistSq
28      rouletteWheel[j] ← totalMinDistSq
29    done
```

```
30    roulettePtr ← randNumGenerator.generateValue() × totalMinDistSq
31    imin ← 1
32    imax ← 1
33    while imax ≥ imin do
34      imid ← imin + floor( ( imax – imin ) / 2 )
35      if imid > 1 then
36        prevRoulettePtr ← rouletteWheel[imid - 1]
37      else
38        prevRoulettePtr ← 0
39      endif
40      if roulettePtr > prevRoulettePtr and roulettePtr ≤ rouletteWheel[imid] then
41        currSeed ← garbageSpots.remove( imid )
42        garbageClusters[i].garbageBinLocation ← currSeed.location
43        break
44      else if roulettePtr > rouletteWheel[imid] then
45        imin ← imid + 1
46      else if roulettePtr < prevRoulettePtr then
47        imax ← imid - 1
48      else
49        throw new Exception( "This case is not reachable!" )
50      endif
51    done
52  done
```

3.1.3.5 Trajectory-based MRTA for Garbage Bin Allocation

The first multi-robot task allocation approach to be considered for solving the garbage bin allocation problem is a trajectory-based approach [5]. In a trajectory-based approach, the optimization process has a search algorithm operate on a single solution per iteration in order to eventually find an optimal solution [5]. Given this premise, the first task will be to define a solution model for the garbage bin allocation problem.

In order to define a solution model for garbage bin allocation, the restrictions imposed by the garbage bin allocation problem are considered. Recalling from Section 3.1.3.2, the only significant restriction imposed by the garbage bin allocation problem is that a 1:1 mapping is enforced between garbage bins and garbage bin locations. That is, each garbage bin location can only be assigned to a single garbage bin at a time. Likewise, each garbage bin can only be situated at a single garbage bin location at a time. Given this restriction, the solution model depicted in Figure 1 and adapted from [5] can be used to represent a possible garbage bin allocation.

p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8
b_3	b_1	b_5	b_4	b_8	b_7	b_2	b_6

Figure 1: A Sample Solution for Trajectory-based Garbage Bin Allocation [5]

As shown in Figure 1, the solution model works as follows. Since the mapping between garbage bins (i.e., the robots) and garbage bin locations (i.e., the tasks) is 1:1, the allocation of garbage bins to garbage bin locations can be represented as an array of garbage bins, where each index in the array corresponds to a garbage bin location and the value at a particular index corresponds to an allocated garbage bin [5]. For

example, suppose that garbage bin b_7 is situated at index 6 of the array. This implies that garbage bin b_7 has been allocated to garbage bin location p_6. As a consequence of this model, each array permutation represents a different allocation of garbage bins to garbage bin locations [5].

Given the solution model described above, it is now possible to solve for a best-effort garbage bin allocation A that minimizes the cost function defined in formula (2) of Section 3.1.2. The fourth year course, ECE 457A Cooperative and Adaptive Algorithms, discusses several trajectory-based search algorithms that are applicable to this case. For the purposes of this design project, the simulated annealing-based multi-robot task allocation used in [5] should suffice (see Algorithm 2 in [5]). For completeness, a slightly modified version of Algorithm 2 in [5] that is adapted for the context of garbage bin allocation is provided below in pseudocode:

Algorithm 6: Pseudocode for Trajectory-based Garbage Bin Allocation Using Simulated Annealing [5]

```
Input Parameters:
   garbageBins:          The set of available garbage bins.
   garbageClusters:      The set of garbage clusters.
   initialTemp:          The initial temperature for simulated annealing.
   finalTemp:            The final temperature for simulated annealing.
   iterationsPerTemp:    The number of iterations per temp value for simulated annealing.
   maxIterations:        The max number of iterations to run simulated annealing for.

Output Parameters:
   optimalBinAlloc:      The lowest cost allocation of garbage bins to garbage bin locations.
   optimalCost:          The cost associated with optimalBinAlloc.

Variable Definitions:
   currentTemp:          The current temperature value for simulated annealing.
   coolingFactor:        The cooling factor for geometric cooling under simulated annealing.
   transitionProb:       The transition probability for a neighbor allocation of higher cost.
   currentBinAlloc:      The current allocation of garbage bins to garbage bin locations.
   neighborBinAlloc:     The neighboring allocation of garbage bins to garbage bin locations.
   currentCost:          The cost associated with currentBinAlloc.
   randNumGenerator:     A random number generator that generates a value between 0 and 1.
   randomNumber:         Stores a randomly generated number between 0 and 1.
   iterationCount:       Keeps track of the current iteration in the algorithm.

Algorithm
   currentBinAlloc ← generateInitialSolution( garbageBins, garbageClusters )
   optimalCost ← currentBinAlloc.getCost()
   currentTemp ← initialTemp
   iterationCount ← 1
   while currentTemp > finalTemp and iterationCount ≤ maxIterations do
     for i ← 1 to iterationsPerTemp do
       neighborBinAlloc ← generateNeighboringBinAlloc( currentBinAlloc )
       neighborCost ← neighborBinAlloc.getCost()
       if neighborCost < currentCost then
         currentBinAlloc ← neighborBinAlloc
         currentCost ← neighborCost
         if neighborCost < optimalCost then
           optimalBinAlloc ← neighborBinAlloc
           optimalCost ← neighborCost
         endif
       else
         randomNumber ← randNumGenerator.generateValue()
         transitionProb ← exp( -1 × ( ( neighborCost - currentCost ) / currentTemp ) )
```

```
35        if transitionProb > randomNumber then
36          currentBinAlloc ← neighborBinAlloc
37          currentCos ← neighborCost
38        endif
39      endif
40    done
41    currentTemp ← currentTemp × coolingFactor
42    iterationCount ← iterationCount + 1
43  done
```

The trajectory-based approach used by Algorithm 6 to arrive at a best-effort garbage bin allocation A works as follows. In the simulated annealing search algorithm, the termination condition is defined using the concept of a cooling schedule [5]. As part of this cooling schedule, the algorithm will track its current temperature (defined in line 9 of Algorithm 6) [5]. Initially, the current temperature will be set for some initial temperature (defined in line 3 of Algorithm 6) such as 100°C [5]. Every few iterations of the algorithm (the number of iterations is defined in line 5 of Algorithm 6), the current temperature will drop by a cooling factor (see line 41 of Algorithm 6) [5]. When the current temperature drops below a certain threshold temperature (defined in line 4 of Algorithm 6) such as 1°C, the algorithm terminates [5]. Additionally, since the number of iterations is to be bounded, Algorithm 6 will also terminate if the iteration count (defined in line 17 of Algorithm 6) exceeds the maximum number of iterations allowed (defined in line 6 of Algorithm 6) [5].

Nested within the cooling schedule for simulated annealing is the optimization algorithm for trajectory-based garbage bin allocation. Before executing the cooling schedule, Algorithm 6 initializes the current garbage bin allocation A for the given set of garbage bins and garbage clusters (the garbage bin locations are a member of each cluster) [5]. Specifically, currentBinAlloc (defined in line 12 of Algorithm 6) is initialized to a garbage bin permutation resembling the one in Figure 1. While executing the cooling schedule, Algorithm 6 will perturb the current garbage bin allocation to obtain a neighboring garbage bin allocation. This is accomplished by randomly performing one of the operations defined in Figure 2 (as adapted from [5]) on currentBinAlloc in order to obtain neighborBinAlloc (defined in line 13 of Algorithm 6). If it turns out that the cost of the neighboring garbage bin allocation is less than the cost of the current garbage bin allocation, Algorithm 6 will assign the neighboring garbage bin allocation as the current garbage bin allocation (see lines 26 to 28 in Algorithm 6) [5]. Otherwise, if it turns out that the neighboring garbage bin allocation is more costly than the current optimal garbage bin allocation, Algorithm 6 will probabilistically select the neighboring garbage bin allocation as the current garbage bin allocation (see lines 33 to 39 in Algorithm 6) [5]. In particular, Algorithm 6 will compute a transition probability and generate a random real number between 0 and 1 inclusive [5]. If the transition probability is greater than this random real number, then Algorithm 6 will assign the more costly garbage bin allocation as the current garbage bin allocation [5]. The purpose of intentionally assigning a more costly neighboring garbage bin allocation as the current garbage bin allocation is to prevent Algorithm 6 from getting stuck in a local minima [5]. When the cooling schedule terminates, optimalBinAlloc (define in line 7 of Algorithm 6) is returned [5].

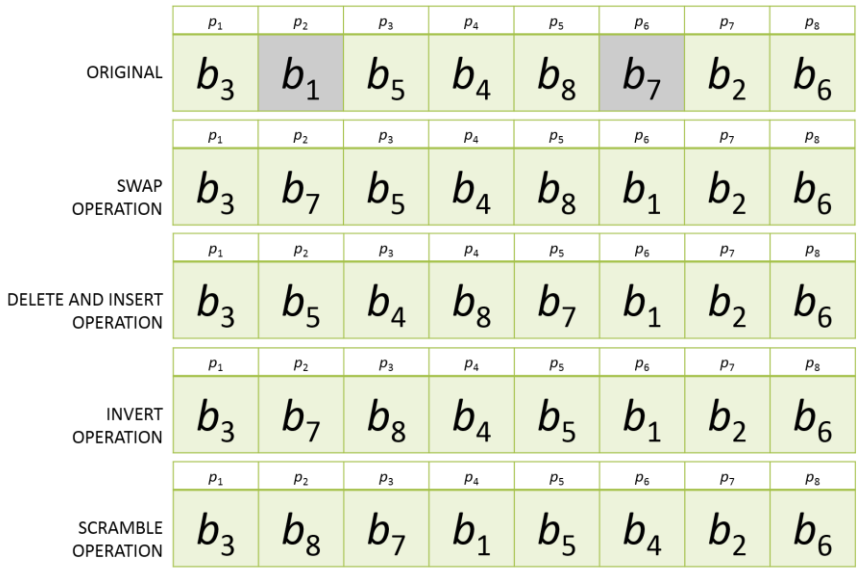


Figure 2: Operations for Generating Neighboring Trajectory-based Garbage Bin Allocation Solutions [5]

3.1.3.6 Simple Auction-based MRTA for Garbage Bin Allocation

The second multi-robot task allocation approach to be considered for solving the garbage bin allocation problem is an approach based on solving the simple auction problem [6]. The objective of this subsection will be to develop a simple auction-based solution model as a possible alternative for solving the garbage bin allocation problem [6].

In the simple auction problem, there exists an auctioneer who is auctioning away a single item and a set of n bidders, each of whom desire to possess this item [6]. Each of the n bidders will submit a bid and the item is sold to the highest bidder [6]. Based on this premise, the server can be analogized as the auctioneer, the available garbage bins as robots can be analogized as bidders, and the garbage bin locations as tasks can be analogized as the items to be auctioned off [6]. Using these simple analogies, the next step is to define a simple auction-based solution model for the garbage bin allocation problem.

To define a simple auction-based solution model for the garbage bin allocation problem, the restrictions imposed by the garbage bin allocation problem are again considered. Given that the garbage bin location problem imposes a 1:1 mapping between the garbage bins and garbage bin locations, an appropriate solution model for simple auction-based garbage bin allocation is a hash table. For this particular hash table solution model, the keys of the hash table would be representative of the garbage bin locations and the values would be representative of the garbage bins. Intuitively, the mapping between a key and a value would represent the allocation of a garbage bin (i.e., the value) to a particular garbage bin location (i.e., the key). A sample garbage bin allocation described in terms of this solution model is illustrated in Figure 3 below. Given the simple auction-based solution model defined here, the following algorithm (i.e., Algorithm 7) partially adapted from [6] can be used to solve for a best-effort garbage bin allocation with respect to the cost function defined in formula (2).

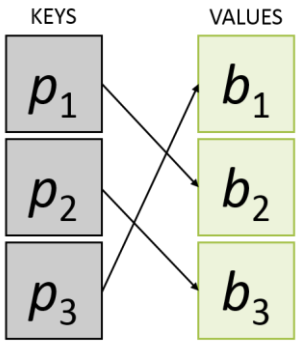


Figure 3: A Sample Solution for Simple Auction-based Garbage Bin Allocation [6]

Algorithm 7: Pseudocode for Simple Auction-based Garbage Bin Allocation [6]

```
Input Parameters:
1   garbageBins:          The set of available garbage bins.
2   garbageClusters:      The set of garbage clusters.

Output Parameters:
3   outputBinAlloc:       A hash table representing allocation of bins to clusters.

Variable Definitions:
4   currentBid:           The current bid by the current garbage bin.
5   lowestBid:            The current lowest bid.
6   lowestBidder:         The current lowest bidding garbage bin.

Algorithm:
7   foreach garbageCluster in garbageClusters do
8     lowestBid ← ∞
9     foreach garbageBin in garbageBins do
10      currentBid ← getBid( garbageBin, garbageCluster )
11      if currentBid < lowestBid then
12        lowestBid ← currentBid
13        lowestBidder ← garbageBin
14      endif
15    done
16    garbageBins.remove( lowestBidder )
17    outputBinAlloc[garbageCluster] ← lowestBidder
18  done
```

The simple auction-based approach used by Algorithm 7 to arrive at a greedy garbage bin allocation A with respect to the cost function defined in formula (2) for the given set of inputs (defined in lines 1 to 2 of Algorithm 7) works as follows. For each garbage cluster C_i in the set of garbage clusters C, an auction is held [6]. The lowest bidding garbage bin with respect to the current garbage cluster is allocated to the garbage cluster (and the corresponding garbage bin location) [6]. Once the auction is over, the resulting garbage bin allocation, outputBinAlloc (defined in line 3 of Algorithm 7) is returned [6].

Given the structure of the simple auction problem, it does not make sense to directly apply the cost function defined in formula (2) for computing the bids made by each of the unallocated garbage bins. This is because unlike the trajectory-based approach for garbage bin allocation defined in Section 3.1.3.5, the simple auction approach defined here is a greedy algorithm that is not concerned with the bigger picture. Rather, the selection of a garbage bin $b_j \in B$ to be allocated to the current garbage bin cluster C_i (and its

corresponding garbage bin location p_i) is largely determined by the individual cost that some mapping of b_j to C_i would contribute to the overall cost as defined by the cost function in formula (2). Hence, the following utility function is instead used to compute the bid:

$$bid(b_j, C_i) = \sqrt{\left(distance(b_j, p_i) + distance(p_i, h_i)\right)^2 + (100(1 - \Pr(\lambda_i \leq v_i)))^2} \quad (7)$$

where $bid(b_j, C_i)$ computes the bid of garbage bin b_j for garbage cluster C_i. λ_i is the average volume of garbage produced in the current hour by garbage cluster C_i. v_i is the the remaining volume in garbage bin b_j. $1 - \Pr(\lambda_i \leq v_i)$ computes the probability that the garbage cluster will overflow within the current hour (see Section 3.1.3 for more details).

3.1.4 Subsystem Data and Analysis

3.1.4.1 Overview

Given the design alternatives for garbage bin location search and garbage bin allocation, the final step in the design iteration is the selection of the design alternatives to be incorporated into the garbage bin allocation subsystem. To meet this objective, the design alternatives for garbage bin location search and garbage bin allocation are validated against the project specifications defined in Section 2. Afterwards, an engineering analysis is carried out to determine the most suitable design alternative for garbage bin location search and the most suitable design alternative for garbage bin allocation.

3.1.4.2 Satisfaction of Project Specifications

For ease of reading, the project specifications from Section 2 that are relevant to the garbage bin allocation subsystem are reproduced below:

1. The cost to be minimized by the garbage bin location search algorithm must account for the distance from the garbage bin location to each garbage spot in the corresponding cluster.
2. The cost to be minimized by the garbage bin allocation algorithm must account for the distance from each garbage bin to its assigned garbage cluster, the distance from each cluster to the nearest service station, and the remaining volume in each garbage bin.
3. The garbage bin allocation subsystem must be able to perform garbage bin allocation on a non-homogeneous set of garbage bins and garbage spots.
4. The garbage bin location search and garbage bin allocation algorithms are best-effort and do not guarantee convergence to the lowest cost solution.
5. Together, the garbage bin location search and garbage bin allocation algorithms must be scalable to at least 1000 garbage spots and 100 garbage bins.
6. For any given garbage bin allocation round, the number of garbage bin locations generated by garbage bin location search must not exceed the number of available bins.
7. Together, the runtime of the garbage bin location search and garbage bin allocation algorithms must not exceed 5 minutes.

Given the design alternatives for the garbage bin allocation subsystem defined in Section 3.1.3, it can be asserted that the project specifications listed above are satisfied. In particular, specification no. 1 and specification no. 2 are satisfied by definition of the cost functions for garbage bin location search (see formula (1) in Section 3.1.2) and garbage bin allocation (see formula (2) in Section 3.1.2) respectively. For specification no. 3, the problem formulation for garbage bin location search and garbage bin allocation in Section 3.1.2 does not assume that set of garbage bins and the set of garbage spots are homogeneous. In particular, the cost functions for garbage bin location search and garbage bin allocation account for the fact that in practice, the average volume of garbage produced in the current hour will differ between garbage spots and the additional volume of garbage that can be contained by a garbage bin in the current hour will differ between garbage bins. For specification no. 4, none of the design alternatives considered for garbage bin location search and garbage bin allocation guarantee convergence to the lowest cost solution. In terms of garbage bin location search, both the k-means and k-means++-based design alternatives are prone to converging to a local minima, which yields suboptimal clusters. In terms of garbage bin allocation, the trajectory-based design alternative uses simulated annealing to optimize garbage bin allocation. Since the runtime of simulated annealing is ultimately bounded by the parameters used to initialize the cooling schedule and not whether the the lowest cost allocation has been found, the yielded garbage bin allocation is best effort. Finally, for specification no. 6, the problem formulation in Section 3.1.2 restricts the quantity of generated garbage clusters to a value less than the number of garbage bins to allow a 1:1 mapping between garbage clusters and garbage bins. Specifications no. 5 and no. 7 will be verified in the next subsection.

3.1.4.3 Simulation and Critical Evaluation

In order to ultimately select the final design for garbage bin location search and the final design for garbage bin allocation, the design alternatives for each category were systematically evaluated against the weighted decision matrix in Table 1 below.

Table 1: The Weighted Decision Matrix for Garbage Bin Allocation Subsystem Design Alternatives

Criteria	Weight	Points	Requirements
Cost	40%	See req.	For garbage bin location search: $points = \begin{cases} \frac{300 - cost}{300} \times 3, & cost \leq 300 \\ 0, & otherwise \end{cases}$
			For garbage bin allocation: $points = \begin{cases} \frac{10000 - cost}{10000} \times 3, & cost \leq 10000 \\ 0, & otherwise \end{cases}$
Runtime	25%	See req.	For garbage bin location search: $points = \begin{cases} \frac{250000 - runtime}{250000} \times 3, & runtime \leq 250000 \\ 0, & otherwise \end{cases}$
			For garbage bin allocation:

			$points = \begin{cases} \frac{250000 - runtime}{250000} \times 3, & runtime \leq 250000 \\ 0, & otherwise \end{cases}$
Runtime Complexity	10%	0	The algorithm tries to solve a problem that has undecidable runtime complexity (i.e., the algorithm might never terminate).
		1	The algorithm is NP-hard or NP-Complete (i.e., the worst case runtime is an exponential function of the number of inputs).
		2	The algorithm has polynomial runtime complexity (i.e., the worst case runtime is a polynomial function of the number of inputs).
		3	The algorithm has linear time complexity or better (i.e., the worst case runtime is at most a linear function of the number of inputs).
Scalability	25%	0	For garbage bin location search, the algorithm is scalable to at most 499 garbage spots.
			For garbage bin allocation, the algorithm is scalable to at most 49 garbage bins.
		1	For garbage bin location search, the algorithm is scalable to between 500 and 749 garbage spots.
			For garbage bin allocation, the algorithm is scalable to between 50 and 74 garbage bins.
		2	For garbage bin location search, the algorithm is scalable to between 750 and 999 garbage spots.
			For garbage bin allocation, the algorithm is scalable to between 75 and 99 garbage spots.
		3	For garbage bin location search, the algorithm is scalable to 1000 or more garbage spots.
			For garbage bin allocation, the algorithm is scalable to 100 or more garbage bins.

Table 1 will evaluate the design alternatives for garbage bin location search and garbage bin allocation against the following criteria: cost, runtime complexity, runtime, and scalability. Cost will evaluate the cost of the solution yielded by each algorithm in their respective category. In general, an algorithm that yields a lower cost will score higher on the cost criterion and vice versa. Runtime will evaluate the time in milliseconds that it takes for each algorithm to run from start to finish during simulation. As with cost, an algorithm that has a shorter runtime will score higher on the cost criterion and vice versa. Runtime complexity evaluates the runtime complexity of each algorithm. Finally, scalability evaluates the scalability of each algorithm in terms of the number of garbage spots for garbage bin location search and the number of garbage bins for garbage bin allocation.

The weightings for each of the criterion are assigned as follows. The cost criterion was assigned the highest weight of 40% because it represents the main objective to be fulfilled by the garbage bin allocation subsystem: to minimize the overhead costs of garbage collection. The runtime criterion and scalability criterion were assigned the second highest weighting of 25% because these criteria are the second most important with respect to the garbage bin allocation subsystem. Algorithms that are scalable to many garbage spots and garbage bins are desirable because it means that the algorithms can adapt to different input sizes. Additionally, algorithms that complete in less time are desirable because it means that the overall system has more time to perform other tasks. Finally, the time complexity criterion was assigned

the lowest weighting of 10% because it is considered the least important criterion. While a simpler algorithm is preferred, cost minimization is more important.

To evaluate the design alternatives for garbage bin location search and garbage bin allocation under the weighted decision matrix presented in Table 1, each design alternative was prototyped in Java 8 and simulated under the same set of randomly generated inputs, where the latitude-longitude position of each input was randomly taken from the infinite set of latitude-longitude positions within a 1km radius of the University of Waterloo. For garbage bin allocation, the design alternatives were also run under the same set of garbage clusters generated from a single run of the k-means++-based garbage bin location search algorithm. The k-means and k-means++-based garbage bin location search algorithms were each executed for 100,000 iterations. Meanwhile, the trajectory-based garbage bin allocation algorithm was executed with a cooling factor of 0.9, an initial temperature of 100°C, a final temperature of 1°C, 100 iterations per temperature instance, and a maximum number of iterations of 1000. The evaluation results for each algorithm against Table 1 are provided in Tables 2, 3, 4, and 5.

Table 2: Evaluation Results for k-means-based Garbage Bin Location Search

Criteria	**Weight**	**Result**	**Points**	**Weighted Score**
Cost	40%	$cost = 196.06273973$	1.03937260	0.41574904
Runtime	25%	$runtime = 627ms$	2.992476	0.748119
Runtime Complexity	10%	$requirement\ 1$	1	0.1
Scalability	25%	$requirement\ 3$	3	0.75
Total				2.01386804

Table 3: Evaluation Results for k-means++-based Garbage Bin Location Search

Criteria	**Weight**	**Result**	**Points**	**Weighted Score**
Cost	40%	$cost = 123.41730848$	1.76582692	0.70633077
Runtime	25%	$runtime = 2216ms$	2.973408	0.743352
Runtime Complexity	10%	$requirement\ 1$	1	0.1
Scalability	25%	$requirement\ 3$	3	0.75
Total				2.29968277

Table 4: Evaluation Results for Trajectory-based Garbage Bin Allocation

Criteria	**Weight**	**Result**	**Points**	**Weighted Score**
Cost	40%	$cost = 8098.20925692$	0.57053722	0.22821489
Runtime	25%	$runtime = 12142ms$	2.854296	0.713574
Runtime Complexity	10%	$requirement\ 1$	1	0.1
Scalability	25%	$requirement\ 3$	3	0.75
Total				1.79178889

Table 5: Evaluation Results for Simple Auction-based Garbage Bin Allocation

Criteria	**Weight**	**Result**	**Points**	**Weighted Score**
Cost	40%	$cost = 8956.72266223$	0.31298320	0.12519328
Runtime	25%	$runtime = 365ms$	2.99562	0.748905

Runtime Complexity	10%	$requirement\ 1$	1	0.1
Scalability	25%	$requirement\ 3$	3	0.75
Total				1.72409828

Based on the results provided in Tables 2, 3, 4, and 5, it can be ascertained that specifications no. 5 and no. 7 are satisfied regardless of which combination of design alternatives is chosen. In this case, the highest weighted score was achieved by k-means++-based garbage location search and trajectory-based garbage bin allocation. Hence, the combination of these two design alternatives are used in the final design of the garbage bin allocation subsystem.

3.2 Network Communications Subsystem

3.2.1 Subsystem Requirements.

The communications subsystem is the component responsible for facilitating communication between a garbage bin and a central system. From the perspective of the communication subsystem, the central system is in charge of storing and possibly displaying information (though the UI and the Database could be split across multiple systems if desired). In practice, the central system will also run the allocation algorithm.

In terms of functional requirements, this subsystem needs to meet the requirement of new garbage bin data being received every 5 minutes and to be able to display this status for each garbage bin as well as alert the users of the user interface to a full garbage bin. The system as a whole must also be scalable (at least 70 garbage bins running at the same time). Even though they aren't listed as specifications, the subsystem as a whole should also adhere to the general engineering principles of simplicity and robustness.

The design of this subsystem is fairly involved in that it requires choosing a network layout, an application layer protocol and an encoding scheme for messages. It also involved designing an application protocol interface, a storage solution for the data and a user interface to display the data.

In the context of this section, communication is done between two entities. Distributed across an area are garbage bins and they are communicating their current status (e.g. storage capacity or battery capacity) with a central system. There may be more than one central system deployed, but in general, a garbage bin is expected to only be communicating with one system at a time (though it could switch systems on the fly if a backup is needed).

An important assumption made in the design of this communication solution is that the garbage bins will accessible by the central system and vice versa. That is to say, the devices will all be on the same subnet (i.e. on the same local network) or the network will be configured in such a way that would allow bidirectional access.

3.2.2 High Level View

The garbage bin collection system uses a server to server (or hybrid as referred to in this report) network layout. A Java Apache Tomcat server is deployed on each central system and a node.js server is

deployed on each garbage bin. The Tomcat server also serves the user interface to any administrators and the allocation algorithm is also run on the central system.

Since servers are deployed on all garbage bins, they are easily reachable by the central system and vice versa as well.

3.2.3 Application Layer Protocol

It was decided that the garbage bins and the central server would communicate using HTTP. HTTP is the foundation for data communication in the World Wide Web [7]. HTTP was chosen because of its wide adoption and simplicity. In addition, there are a large number of servers readily available that communicate with HTTP.

HTTP defines 9 different request methods [7]. Of the 9 methods HTTP defines (GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH), the only two used in this garbage collection system are GET and POST. GET is used to retrieve information (such as the garbage bin's last known status) and POST is used to request an update on the bin's status and used by the garbage bin to send an update to the central system.

3.2.4 Message Encoding

HTML was chosen as the protocol used to transfer data, but the data itself also needs to be encoded (or formatted) in a manner that is understood by both the central system and the garbage bins. The format of the data needs to be simple, easily extensible (so more data can be added down the line) and it should be easy to work with.

One option for message encoding is to use a custom String (or Binary) format (e.g. 78|89|94|23 where the first number could be a latitude, the second a longitude, the third capacity and the final battery.). This approach is fairly simple, but it is not easily extensible (adding to the format could break the existing parsers) and nor is it easy to work with because formatting has to be exactly adhered to or lots of complexity gets added to be added to the parser.

Another frequently used document encoding scheme is Extensible Markup Language (XML). XML, like HTML, uses tags to denote elements that make up the data [8]. The main disadvantage with XML is that it is verbose and the structure is fairly rigid and difficult to modify down the line.

The last option considered was JSON. JSON stands for JavaScript Object Notation and it is a lightweight data-interchange format that is easy for both humans and machines to work with [9]. JSON was the interchange (or message encoding) option decided upon for use in this project.

3.2.5 Application Protocol Interface

To facilitate communication between the central system and the garbage bins, two application protocol interfaces (API) were designed and implemented. The HTTP server running atop the garbage bins must provide certain GET and PUT methods that can be accessed by the central server when the central server desires an update. Similarly, the central server must implement its own set of GET and PUT methods that can be accessed by the garbage bins when they are sending data over. Recall that data transfer is not

synchronous. The garbage bins measure data after they have been asked to do so and they only send it once they have measured it.

3.2.5.1 Garbage Bin API

The Garbage Bin API is the API the server running atop the garbage bin must conform to. All methods that the central server would use on a garbage bin require no input parameters and return JSON objects containing information about the garbage bin. The important functions are 'lastStatus' and 'updateStatus'. 'lastStatus' simply returns the last readings the garbage bin did formatted as a JSON object. 'updateStatus' reminds the garbage bin to update its current status. Once the garbage bin has updated itself, it posts the new status to the central server at its own discretion.

'updateStatus' makes an asynchronous command line call. It executes an executable that activates the pins to interface with the ATMEGA chip and its sensors in order to acquire a new reading. This call to the external executable should never result in two instances of the sensing executable running at once. If two instances try to interface with the ATMEGA chip at the same time, the pins could be activated incorrectly and this could result in incorrect values, crashed software or deadlocked software. These external calls could end up made concurrently if 'updateStatus' is called multiple times in rapid succession.

To prevent this from happening, 'updateStatus' uses a semaphore to restrict access to the command line. This semaphore is acquired right before the external call is made and is released right after it is finished.

3.2.5.2 Central System API

The HTTP server running atop the Central System needs to provide an API to the distributed garbage bins in order to successfully receive information from them. Some of the API calls are also used by the User Interface to display data and in theory could be used in a distributed system to share data across multiple central systems. The most important function is called 'postNewStatus' and it is used by the garbage bins to asynchronously send their updated status to the central server. Another important function is 'registerBin' which allows a garbage bin to notify a server of its existence when the bin first boots up.

3.2.6 Data Flow

There are two separate data flows in this system. One dataflow occurs when a garbage bin first starts. The other dataflow is the steady state dataflow that occurs every couple of minutes, usually 3, where the central system polls the garbage bin for an updated status.

For the sake of simplicity, the central system does not keep a separate data store for garbage bins that are registered but haven't updated their status yet. So, right after responding to a bin's register request, the central system immediately makes a 'lastStatus' call to the garbage bin. The returned status allows the central system to add an entry into its garbage bin status map. This imposes a very simple restriction on the garbage bins. They must update their status at least once of their own accord right after booting up and before registering. The data flow for the registration process is illustrated in Figure 4.

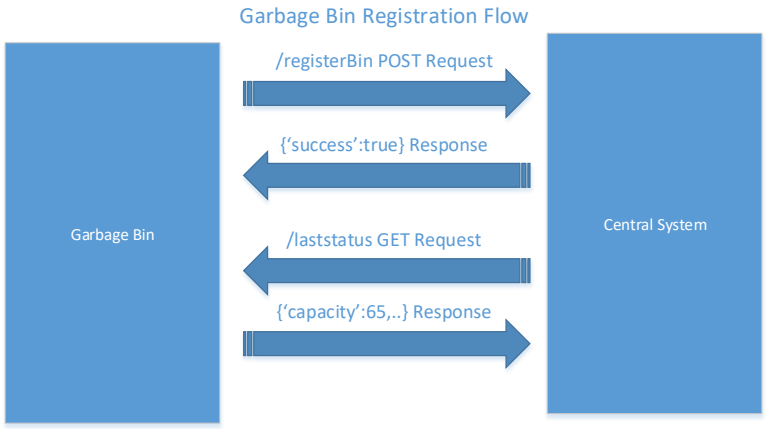


Figure 4: Garbage Bin Registration Data Flow

When the system is in a steady state (i.e. registration has already been done), then data is exchanged asynchronously. Every couple of minutes, the central system will make a request to the garbage bin to update its status. The garbage bin will begin updating its status and once it is complete, it will respond with a POST request of its own. This is illustrated in Figure 5.

Naturally, the central system will do this for all garbage bins registered to it. It does this by periodically iterating through its in memory data store of statuses of various garbage bins and compares the current time against the timestamp that arrived with the status. If the timestamp on the status is more than 3 minutes old, the central system will make an 'updateStatus' POST request. This is done for every outdated status until the maximum user defined threshold for concurrent garbage bin updates is reached at which point the server will wait for a while before polling once more.

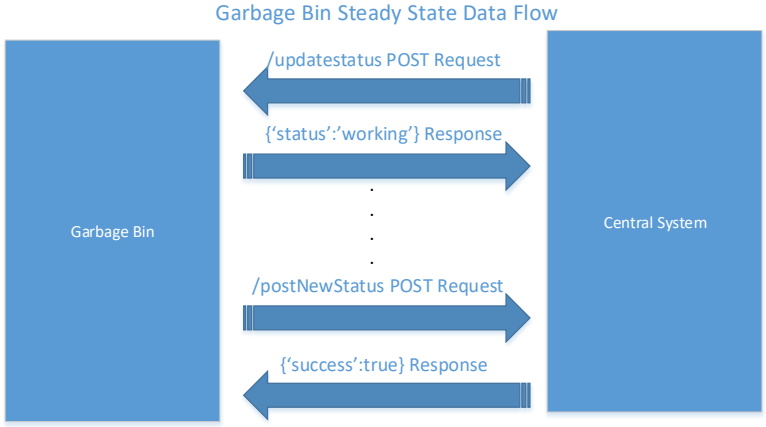


Figure 5: Data Flow in a Steady State System.

3.2.7 Data Storage

Given that the data sent back by garbage bins is fairly simple and that there aren't any long lived queries performed against it, the live garbage bin status data is simply kept in memory as Java objects. For maximum interoperability with the API calls, the Java representation of the data is encapsulated in a class that provides functions to convert it back to a Java JSON object or straight to a JSON String (only the String representation of a JSON object can be sent over HTTP).

The central system maintains a global synchronized map that maps a bin ID to its corresponding status. This mapping is updated every time the central system receives a new status update from the garbage bin. That is to say, no history is kept in memory. This allows for fast retrieval of data by the UI and for fast updates of data whenever a garbage bin sends an update. It could also facilitate easy sharing of data in between central systems should a more distributed approach be required, but this is out of the scope of this project.

Finally, the Java objects that remain in memory are periodically written out to a MySQL database, so that they may be stored for later use for statistical analysis and trend observation.

3.2.8 Data Display (User Interface)

The User Interface needs to display the status of all garbage bins deployed in the field and it needs to highlight any problematic bins (bins that are full or can't be reached). The UI should also allow for requesting an update of a garbage bin's status. The UI displays a table populated via a /garbageBins GET call to the central server. This table contains all the garbage bins the central server knows about and a refresh button is provided to request an update for a given bin as well. This update triggers a POST call to the selected garbage bin without going through the backend. Once the garbage bin is updated, it will automatically update the Central Server as well. The UI auto refreshes every minute or so and this refresh will display any new information that was received.

Table 6: User response to delays [10]

Delay	Effect
0.1 seconds	Feeling of instantaneous response, the outcome feels like it was caused by the user, not the computer.
1 second	Keeps the user's flow of thought seamless, the delay is felt, thus the user knows the outcome is generated by the computer, but overall they still feel in control
1-10 seconds	Keeps the user's attention, however the user starts to think about other things
10 seconds	Will often make the users leave the site

The UI was written with response times in mind. According to the study done by Jakob Nielsen as seen in table above, there are negative consequences due to poor response times as seen in table above. As a result, in order to keep the operator engaged in the web application, the response times should be under one second. There are several factors which will affect this time as learned from Networking ECE 358 and ECE 416. Wireless communication (Wi-Fi) has a high Bit Error Rate. As a result, multiple retransmissions will be done by TCP in order to correctly receive and deliver packets. This means that processing time must compensate for the delay. As studied in Algorithm Design ECE 406, processing time is balanced out by memory usage. If more pre-processing is done on data, and is stored, it will reduce the real-time processing cost. What this means for this project, is that more tables will be needed in the database.

Instead of having just two database tables, with general data, and performing queries on that data, the result from queries should be put into a new table so that the data only needs to be returned to the user

without processing overhead. Principles from UI course CS 349 were also applied when designing the web application

3.2.9 Design Iterations and Alternatives

3.2.9.1 Network Layout

In the context of a network communication subsystem, network layout refers to the relationship between devices on a network. In designing a layout, a solution to questions such as, who initiates conversation, how tasks are delegated and how are tasks shared are desired to be found. There are two main network layouts in use, the Client/Server model and the peer-to-peer model.

The traditional network model is called the Client/Server model where a single, central host communicates with several clients. In this model, the clients always initiate conversation as their location is generally not known to the server. The workload distribution is not symmetric in this model. That is to say, the server will perform different tasks then the clients that talk to it. On the other hand, there is the peer-to-peer model. In a peer-to-peer network, tasks are split across nodes and there is no central server to coordinate. For the Garbage Collection System, a pure peer-to-peer approach is not ideal as a user interface is required to manage clients. The allocation of garbage bins needs to be done from a central system and the computation for it cannot be split easily across multiple systems.

Initially, the network paradigm was one server, many clients. The garbage bins would connect to the server and send it data every time a command line program was run. The data was also sent as a simple string using a delimiter to separate out the values. This earliest iteration also transmitted data over a simple UDP connection.

The primary advantage of this approach is simplicity. There is only one server and the communication between the garbage bins and the server is fairly simple. This approach could work well, but it is not very robust and it is not as scalable as it could be. By sheer coincidence, all of the garbage bins could contact the central system at the exact same time. If the central system happened to be busy with other work (serving the UI to many administrators, or running the allocation algorithm), this could cause a network traffic if there were a large number of garbage bins. That is to say, the garbage bins are simply sending data when they are programmed to do so. The data flow cannot be easily controlled or regulated by the central system which might desire to only receive data when it is convenient to do so.

Further harming robustness and ease of administration is that the central system knows nothing about the garbage bins until they contact it. There is no way of doing a live check on the garbage bins. That is to say, a garbage bin could have a problem and the central system (and any users using the system) would not know until the server decides that the garbage bin was taking too long before sending new data. This requirement for additional monitoring adds complexity and reduces the simplicity gained by having one contact point.

Another approach considered was having the garbage bins act as servers and the central system as a client. In this particular the garbage bins are periodically contacted by the central system. This approach's primary advantage is that the central system can decide when it needs the data, so if it is bogged down with other work, it can simply wait until it has the processing power. More importantly, it can stagnate the data collection over time, so data is never received at the same time.

The primary disadvantage of this approach is that the central system needs to know the IP address (or equivalently a hostname that can be resolved) for every garbage bin before initial contact can be made. That is to say, an administrator needs to manually enter the address for each garbage bin after it gets deployed and while this is doable for a small number of bins, it is far more difficult for a faster moving environment with a greater number of bins.

A third option for the garbage bin collection system is a hybrid solution. That is to say, the garbage bins and the central system act as both servers and clients. In this paradigm, the garbage bins know of the IP address of the central system and use it to register with the central system. The server running atop the central system then records the IP address of the garbage bin and from then on periodically contacts the garbage bins.

An administrator does not need to input the IP address of the garbage bin ahead of time as the garbage bin will, on its own accord, contact the central system when it starts up. Unlike the layout with a single server, the central system (once a garbage bin has been registered) can contact the garbage bins at its own discretion and thus pace out the influx of garbage bin data.

In addition, this paradigm facilitates asynchronous messaging. Since contact can be made in a bidirectional manner, the garbage bins don't have to send data right away. In the garbage collection system, the central system contacts a given garbage bin and asks the bin to update its status. The garbage bin performs sensor readings, tabulates its information and then sends the information to the central system when it is ready to do so.

Unlike the previous two network layouts, the central system does not need to wait for this information to be ready. It can go and contact other garbage bins or the central system can perform some other work. This can lead to a great increase in efficiency if the central system has a large amount of work to do (i.e. it serves a UI to several administrators and/or it is frequently running the allocation algorithm). This data flow was discussed and illustrated in Section 3.2.6.

It is important to note that all three network layouts can meet the functional specifications required of them. All of them can scale up to 70 garbage bins and all of them can meet the maximum 5 minute update rate. That said, they are not all equal when it comes to criteria sought after.

The criteria sought in the network layout are simplicity, scalability, robustness and ease of administration. The weights these criteria were assigned are shown in Table 7.

Table 7: Assigned Weights for the System's Network Layout

Criteria	Simplicity	Scalability	Robustness	Administration
Weight	20%	30%	30%	20%

Simplicity is always important in engineering design as simple designs break less often and are less prone to hidden problems. Simplicity is difficult to measure quantitatively, so a score was assigned on an estimate of how much complexity would have to be added to make the solution work well. The scores for simplicity are shown in Table 8.

Table 8: Scores for Simplicity

Criteria	Central Server	Bins are Servers	Hybrid
Simplicity	$\frac{7}{10} * 20 = 14$	$\frac{6}{10} * 20 = 12$	$\frac{8}{10} * 20 = 16$

Having the central system be the lone server is a fairly simple solution as communication is easy, but the addition of monitoring (to check when bins are down) adds a bit of complexity and brings the score down.

Having every bin be a server removes the need for additional monitoring, but it requires the deployment of many servers, and prepopulated additional state to keep track of every garbage bin is deployed. This additional state hurts simplicity.

The hybrid solution does not require advanced monitoring and while it does keep state of every bin that is deployed, this state is easy to auto populate because the garbage bin's register with the central system. The hybrid solution is also simple in that it can facilitate total asynchronous communication, so connections to garbage bins don't need to be long lived and there is no need to wait for data to be ready (as it is only sent when it is ready).

Scalability is very important because it is listed as a required spec. Even if the requirement of 70 garbage bins is met, more scalability is good as it allows the project to grow in the future. The scores for scalability testing are shown in Table 9.

Table 9: Scores for Scalability

Criteria	Central Server	Bins are Servers	Hybrid
Scalability	$\frac{7}{10} * 30 = 21$	$\frac{7}{10} * 30 = 21$	$\frac{8}{10} * 30 = 24$

Scalability is also difficult to properly test without the entire system in place, and we don't have the resources to create 70 or more prototypes. A simulation test was tried wherein a single call to the central system spawned 300 threads (simulates the threads created when a web server receives requests from 300 discrete calls), but no measurable impact on CPU usage was found.

All of the options (since all of them can request or receive data from that many garbage cans) are fairly scalable. The Central Server approach loses out a little because it actually does have to deal with bursts and while 300 garbage bins are shown to be fine, more might cause problems. Having the bins as servers loses out a little because state tracking what bins are deployed has to be maintained somewhere and this can become difficult as the number increases (generally requires human input).

The hybrid solution is especially scalable not only because it can contact the bins of its own accord, but it can close the connection and simply wait for them to contact it back.

Robustness is just as important as scalability. The system as a whole should be fairly resilient and should report problems as they show up. This is also hard to measure without having a system deployed for a period of time, but an effort was made to reason an acceptable robustness score for each option. Scores for robustness are shown in Table 10.

Table 10: Scores for Robustness

Criteria	Central Server	Bins are Servers	Hybrid
Robustness	$\frac{5}{10} * 30 = 15$	$\frac{7}{10} * 30 = 21$	$\frac{8}{10} * 30 = 24$

When the central system is the only server, the garbage bins cannot be polled to get their status, so the central system is in the dark when it comes to their actual health or at the least to the cause of their problems.

Having a server deployed on the garbage bins allows for greater insight into any problems and better problem resolution, but since the garbage bins have to measure their status when the central system contacts them, the HTTP connection is open for longer and a connection that is open for longer is more susceptible to being dropped or broken. As I learned in ECE 358 and ECE 416, the underlying network used for communication makes no guarantees about quality even more so if wireless networks (as might be the case for connected garbage bins) get involved.

The hybrid solution is very robust because the garbage bins can be easily monitored and easily debugged when they have a problem. Since the hybrid solution can be implemented with asynchronous network calls, the connections can be open for really short amounts of time, so they are less susceptible to being broken. If the method calls (the API) are well designed, there is very little to no state kept as well. This allows for a very simple, but also very robust system.

Ease of administration is based on the options a UI could give to the administrator. The three layouts are ranked in Table 11.

Table 11: Scores for Administrative Ease

Criteria	Central Server	Bins are Servers	Hybrid
Administrative Ease	$\frac{7}{10} * 20 = 14$	$\frac{4}{10} * 20 = 8$	$\frac{8}{10} * 20 = 16$

If the central system is the only server, the UI can only display the last known status of each garbage bin and alert the user to any bins that did not make contact as well as any bins that appear full.

In either of the other two options, the UI could tap directly into the garbage bin's endpoints and provide up to date information when the administrator requests it. Having only bins as servers will however greatly increase administrative load since the IP address of a garbage bin has to be manually entered or activated every time a bin is deployed. The hybrid solution can get around this by enforcing garbage bin registration, so it received the highest score.

Table 12 lists the individual criteria scores and sums them up. All three solutions can be made to meet the required specifications, but out of the three network layouts, the hybrid solution got the highest score, so it was decided that this is what would be implemented and used for the Garbage Bin Collection System.

Table 12: Totals for Network Layouts

Criteria	Central Server	Bins are Servers	Hybrid
Simplicity	$\frac{7}{10} * 20 = 14$	$\frac{6}{10} * 20 = 12$	$\frac{8}{10} * 20 = 16$
Scalability	$\frac{7}{10} * 30 = 21$	$\frac{7}{10} * 30 = 21$	$\frac{8}{10} * 30 = 24$
Robustness	$\frac{5}{10} * 30 = 15$	$\frac{7}{10} * 30 = 21$	$\frac{8}{10} * 30 = 24$
Administrative Ease	$\frac{7}{10} * 20 = 14$	$\frac{4}{10} * 20 = 8$	$\frac{8}{10} * 20 = 16$
Total	64/100	62/100	80/100

3.2.9.2 Garbage Bin Server

After deciding that the garbage bin would run a server, Python was initially chosen as the language of choice due to its elegance and built in functionality. Some Python code was written for this project, but after further discussion, it was noted that node.js was a better fit. Node.js is itself very functional and libraries like express allow for the creation of powerful, robust web servers. In addition, using node.js eased communication because of it's out of the box asynchronous nature.

3.2.9.3 Design Alternatives Considered For the UI

The UI subsystem has a very specific purpose of allowing the operator to interface with the field hardware from a distance. One alternative that the group has considered was to display metrics about a specific garbage bin on an LCD display. This has an advantage of showing information at the specific location, but does not contribute to the allocation algorithm.

Furthermore, an allocation algorithm needs to run on some platform, and a web application doubles as that. Without the web application, the system would be more difficult to test for the developers. The fact that the interface needs to be accessible puts certain constraints onto the UI application. One such constraint is that any general user must be able to view the information. There is no better way to do that than to use a web application due to abundant usage of web browsers.

In essence, a native application would take more time to develop, and would not be modular in its usage because every user needs to install it. However with a web application, it just needs to run on the central server and users can access it from their phone or desktop.

Table 13: UI Alternatives

Suggestion	**Web Application**	**Native Application**	**LCD Status**
Feature Set	3	3	1
Modularity	3	2	1
Development Time	2	3	2
Requires Internet	1	1	3
Usefulness	3	2	1

Total	12	11	8

As seen in Table 13, the Web Application solution is the most feasible choice for our purposes. With many of the group members having done that specific area of development over co-op terms, it should further be stated that this specific area has the most knowledge out of all the alternatives.

Furthermore, the Web application satisfies functional spec 4 which requires displaying all garbage bins currently in the field. This spec could not be satisfied with a simple LCD Status. Lastly, in order to exclude the bins from the allocation algorithm, it would need to be closely connected with that subsystem. The two are implemented in the same web application in order to satisfy functional spec 6.

3.2.10 Testing and Analysis

In addition to the analysis done in Section 3.2.9.1, tests were run against the designed solution to try and ensure correctness.

To try and emulate the impact of many garbage bins contacting the central server at once, 300 worker threads were spawned and each one was delegated the same amount of work a single POST request received from a garbage bin would be expected to do. The CPU usage only jumped up by around 0.1%, so the impact of making this many short lived threads was very minimal on the central system. This test was run on a Sony Vaio laptop with an Intel Core i5 CPU. A production ready server would be expected to have better hardware and thus the impact would be even less.

It was also decided that the stability of the 'updateStatus' call (on the garbage bin) would be tested. This call was tested to make sure it wouldn't create more than one instance of the command line executable. Instead of executing the sensor reading code, the node.js server would execute a simple python script that could detect if any other identical scripts were running (by checking if a certain file existed).

100 concurrent 'updateStatus' calls were made to the garbage bin node.js server and the semaphore held. The command line calls were made serially and not concurrently. This proved that even in the unlikely case that the garbage bin is contacted by multiple devices at the same time, it will still hold and won't collapse. This helped prove the robustness and resilience of our system.

Given that the system can be seen to tolerate over 300 concurrent garbage bin calls, it can be said to scale up to the minimum requirement of 70 garbage bins. When a garbage bin updates its status, it may take at most 30 seconds (in the case that the GPS module cannot get a fix).

Given a maximum network delay of 1 second (this is very high and it is actually likely to be significantly less than 100 ms), it can be said that a garbage bin will update the central system at most 31 seconds after receiving the request.

Dividing our 5 minute requirement (every garbage bin must update its status at most every 5 minutes) into blocks, we get 300 seconds / 31 = 9.6774 units of time. If we then divide the minimum supported garbage bin number of 70 by 9.6774, we get 7.23 or rounding up, 8 garbage bins, per time block.

That is to say, at the start of every 31 second time slice, the central server needs to be able to contact 8 garbage bins and it should be able to tolerate the data they send during (or in the worst case at the end of) that 31 second time slice. Given that the system has been tested with up to 300 concurrent garbage bins, 8 concurrent garbage bins should pose no problem.

All 70 garbage bins could be scheduled to update at roughly the same time and the system would be able to handle it. That said, knowledge gleamed from ECE 416 suggests that wireless networks are very prone to interference and congestion, so depending on the network infrastructure present in the deployment zone, it might be wise to not contact all the garbage bins at once.

3.2.11 Data Usage Analysis

The data that gets sent from the garbage bin includes the following information about the garbage bin: {Bin ID, Location, Current Capacity, Timestamp}. When the capacity received is between 90%-100% then the UI will use this information to indicate that the garbage bin is full. This satisfies functional spec 5.

One extremely important usage parameter would be the busyness factor of the selected garbage bin. The statistic that lets the operator know if the bin is busy, is if its capacity fluctuates quickly in a fixed period of time. This would be the rate of change of capacity with respect to time:

$$\frac{dC}{dt} = \frac{\Delta c}{\Delta t} = \frac{c2 - c1}{t2 - t1}$$

The higher the rate of change of capacity is, the busier that location is. Since the garbage bin status data is sent over a regular time interval (i.e. 5 minutes), then the difference in time remains constant, while difference in capacity may fluctuate.

In order to make this useful to the operator, the table must display the busiest bins not at every time interval, but over the entire day. The individual 5 minute interval data could be plotted to give visual representation, but the entire day data will give some insight into the future. For this to work, each bin must have a busyness factor associated with it to allow for sorting by busyness. This can be done by the summation of all the rates available throughout the day. The idea is, the higher rate will yield a higher total sum:

$$\sum_{i=0}^{n} \frac{dC_i}{dt_i} = \sum_{i=0}^{n-1} \frac{c_{i+1} - c_i}{t_{i+1} - t_i}$$

Below the data is provided for meeting functional specs:

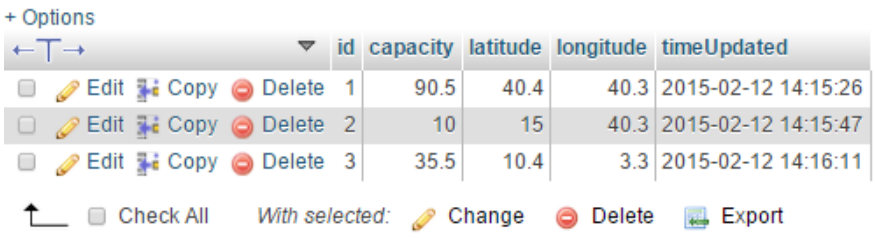


Figure 6: Functional Spec 4

The above data is for functional spec 4, which shows the capacity and the id of each garbage bin. This table is called GarbageBinLog.

For functional spec 5, a simple query such as: "`SELECT * FROM` `garbagebinlog` `WHERE capacity>90`" Is run on the table above to extract only nearly full garbage bins form the table resulting in:

+ Options

	id	capacity	latitude	longitude	timeUpdated
Edit Copy Delete	1	90.5	40.4	40.3	2015-02-12 14:15:26
Edit Copy Delete	3	100	3	75	2015-02-12 14:29:57

Figure 7: Functional Spec 5

For functional spec 6, an opposite query from functional spec 5 is used to extract all the non-full garbage bins: "`SELECT * FROM `garbagebinlog` WHERE capacity<90`" The above query results in the list of garbage bins which should be included in the allocation algorithm

+ Options

	id	capacity	latitude	longitude	timeUpdated
Edit Copy Delete	2	10	15	40.3	2015-02-12 14:15:47
Edit Copy Delete	3	35.5	10.4	3.3	2015-02-12 14:16:11

Figure 8: Functional Spec 6

3.3 Garbage Bin Sensors Subsystem

3.3.1 Subsystem Overview

The third subsystem developed as part of this project is the garbage bin sensors subsystem. As a whole, the fundamental objective of this subsystem is to serve as a peripheral for collecting data on a garbage bin. The data that is collected by this subsystem includes the location of the garbage bin and the container depth of the garbage bin. Using this data, several important functions can be realized. In particular, the container depth can be used by the server to approximate the volume of empty container space in the garbage bin. With this approximation for the volume of empty container space, the server can then infer whether the garbage bin needs to be collected or reallocated. When used in tandem with the garbage bin's location data, the server can additionally learn about the production patterns of garbage in the area by hour. This enables the server to make more informed decisions with regards to garbage bin allocation as more data is collected over time.

3.3.2 Design Description

The design of this subsystem focuses on the integration of the hardware modules used for collecting data (i.e., location and container depth) on the garbage bin, as well as the development of the embedded software that drives the integrated hardware as a whole.

3.3.2.1 Integrated Hardware

The integrated hardware for the garbage bin sensors subsystem includes the following hardware modules: the HC-SR04 ultrasonic ranging module [11], the TMP36GZ temperature sensor [12], the Adafruit Ultimate GPS Breakout v3 module, the LCM1602C 16x2 LCD module, and the Atmel AVR ATMEGA328P-PU microcontroller (MCU) [13]. Each hardware module fulfills a specific purpose with regards to collecting data on the garbage bin. The HC-SR04 ultrasonic ranging module serves as a sensor for measuring the container depth of the garbage bin. The TMP36GZ temperature sensor serves as a sensor for measuring

the ambient temperature of the garbage bin[1]. The Adafruit Ultimate GPS Breakout v3 module serves as a sensor for obtaining the current location of the garbage bin in terms of latitude and longitude coordinates. The LCM1602C 16x2 LCD module serves as a display for the data collected by the garbage bin. Finally, the Atmel AVR ATMEGA328P-PU MCU serves as the brain and heart of the garbage bin sensors subsystem and is responsible for driving the aforementioned hardware modules to perform their respective functions.

Given these hardware modules, the integrated hardware design specified by the schematic in Figure 4 is implemented for the garbage bin sensors subsystem. As shown in Figure 4, this design is centered on an Atmel AVR ATMEGA328P-PU MCU clocked at 16MHz using an external crystal oscillator connected between pins PB6 (i.e., pin 9) and PB7 (i.e., pin 10) [13]. Branching from this MCU are the rest of the hardware modules, each of which are strategically connected to specific pins on the Atmel AVR ATMEGA328P-PU MCU so as to support specific operations on the software end with respect to each hardware module. In particular, the ECHO pin on the HC-SR04 ultrasonic ranging module is connected to pin PB0 (i.e., pin 14) to support the use of the input capture interrupt on the Atmel AVR ATMEGA328P-PU MCU for recognizing input pulses from the HC-SR04 ultrasonic ranging module [13]. Meanwhile, the V_{out} pin on the TMP36GZ temperature sensor is connected to pin PC0 (i.e., pin 23) to support the use of the analog-to-digital convert (ADC) on the Atmel AVR ATMEGA328P-PU MCU for receiving analog input from the TMP36GZ temperature sensor [13]. Finally, the TX pin on the Adafruit Ultimate GPS Breakout v3 module is connected to pin PD0 (i.e., pin 2) to support the use of the universal asynchronous receiver/transmitter (UART) on the Atmel AVR ATMEGA328P-PU MCU for receiving a continuous stream of serial data from the Adafruit Ultimate GPS Breakout v3 module [13].

[1]The ambient temperature is required to approximate the speed of sound, which in turn is required for interpreting depth from the input pulses sent by the HC-SR04 ultrasonic ranging module [11].

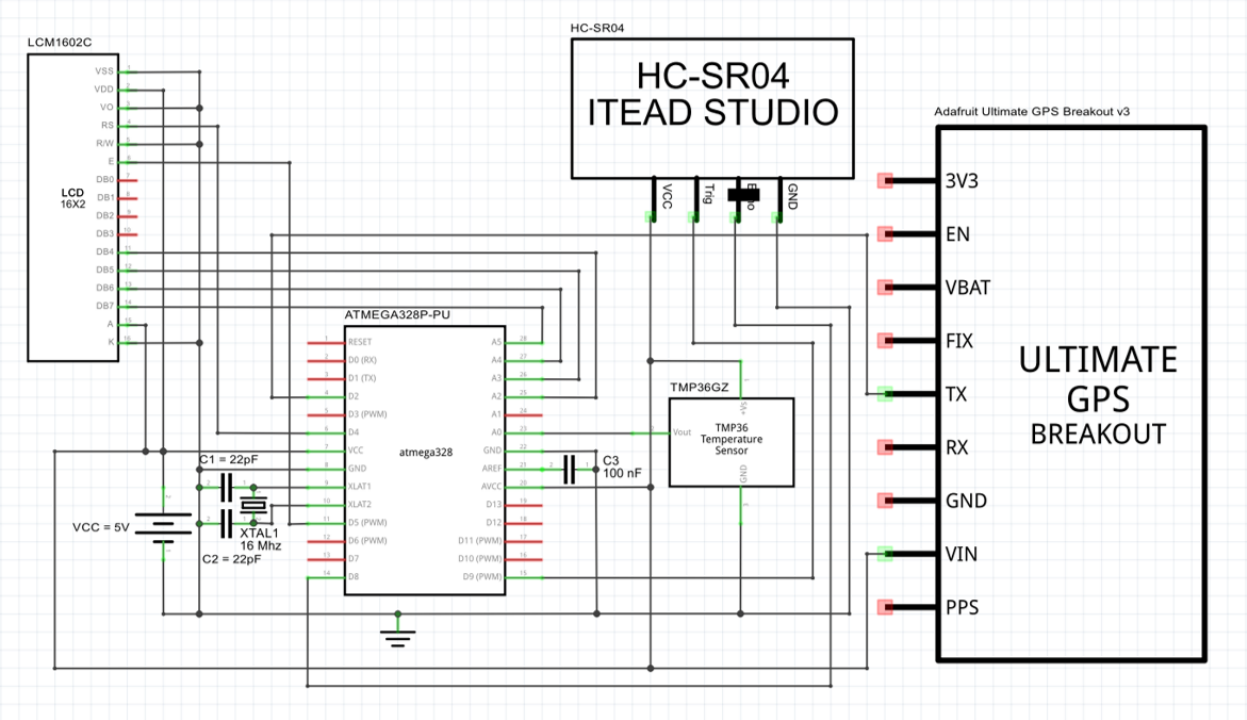


Figure 9: The Integrated Hardware Schematic for the Garbage Bin Sensors Subsystem

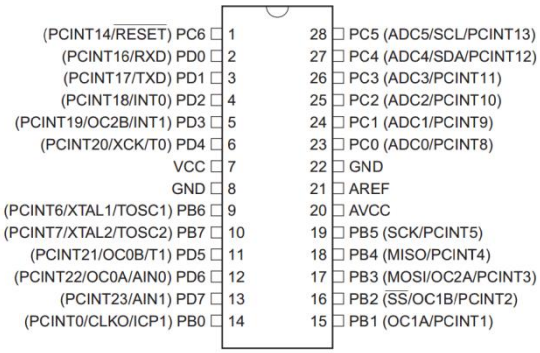


Figure 10: The Pinout for the Atmel AVR ATMEGA328P-PU MCU [13]

From this design, the final implementation of the integrated hardware is intended to be fit into an enclosure. This enclosure, in turn, will be positioned centrally to the garbage bin container lid. To facilitate the measurement of the garbage bin's container depth, the HC-SR04 ultrasonic ranging module is positioned such that it is facing towards the bottom of the garbage bin from inside the enclosure.

3.3.2.2 Embedded Software Architecture

The embedded software that drives the integrated hardware for the garbage bin sensors subsystem is designed around an asynchronous master/slave model for data collection. The idea here is that the garbage bin sensors subsystem would fulfill the role of a slave when prompted by some external master, which for the purposes of this project is the Raspberry Pi Model B+ single board computer. What this

model entails for the garbage bin sensors subsystem is that it should only collect data on the garbage bin's ambient temperature, container depth, and location when prompted by the Raspberry Pi.

To implement this asynchronous master/slave model, the embedded software is structured as follows. In the main program, the embedded software first initializes the hardware drivers for each hardware module (e.g., the LCM1602C 16x2 LCD module, the TMP36GZ temperature sensor, the HC-SR04 ultrasonic ranging module, and the Adafruit GPS Ultimate Breakout v3 module). The main program then enters an infinite loop[2]. Within this infinite loop, the main program will persistently check whether the master has made a request for data collection. This condition is represented as an update flag. If it turns out that this update flag has been set, the main program will then proceed to call the TMP36GZ temperature sensor driver, the HC-SR04 ultrasonic ranging module driver, the Adafruit GPS Ultimate Breakout v3 module driver, and the LCM1602C 16x2 LCD module to collect and display the updated data pertaining to the garbage bin's ambient temperature, container depth, and location. The collected garbage bin data is then sent to the master from pin PD1 (i.e., pin 3) using the UART on the Atmel ATMEGA328P-PU MCU, whom in turn relays the collected data to the server. Finally, the main program clears the update flag and waits for the master's next data collection request.

Diving deeper into the implementation of this asynchronous master/slave model, the embedded software implements the master data collection request as an external rising edge-triggered interrupt. Under this implementation, the master sends a data collection request to the integrated hardware by externally driving high pin PD2 (i.e., pin 4) on the Atmel AVR ATMEGA328P-PU MCU [13]. The Atmel AVR ATMEGA328P-PU MCU, detecting the rising edge on pin PD2 (i.e., pin 4), then proceeds to perform a context switch in order to transfer control from the main program over to the external interrupt service routine for pin PD2 (i.e., pin 4) [13]. This ISR then sets the update flag to indicate that a data collection request has been made by the master before a second context switch is performed to return control from the ISR to the main program [13]. Upon regaining control, the main program recognizes that the update flag has been set and proceeds accordingly.

Finally, to support the implementation of the asynchronous master/slave model described in this subsection, the following additional connections are added on the hardware end (see Figure 12 below). First, an additional connection is made from the general purpose input/output (GPIO) pin 17 on the Raspberry Pi Model B+ to pin PD2 (i.e., pin 4) on the Atmel AVR ATMEGA328P-PU MCU. Using this connection, the Raspberry Pi Model B+, as the master, will be able to drive pin PD2 (i.e., pin 4) on the Atmel AVR ATMEGA328P-PU MCU high in order to make a request for data collection [13]. Second, an additional connection is made from pin PD1 (i.e., pin 3) on the Atmel AVR ATMEGA328P-PU, passing through a Sparkfun Bidirectional Logic Level Converter module[3], to GPIO pin 15 on the Raspberry Pi Model B+ [13]. Using this connection, the garbage bin sensors subsystem, as the slave, will be able to use the

[2] The main program will stay in this infinite loop for the duration in which the integrated hardware is powered on.
[3] The Sparkfun Bidirectional Logic Level Converter is used to convert the serial data transmitted using the UART on the Atmel AVR ATMEGA328P-PU MCU from a 5V signal to a 3.3V signal, as the GPIO pins onboard the Raspberry Pi Model B+ are unable to handle voltages above 3.3V.

UART on the Atmel AVR ATMEGA328P-PU MCU for transmitting the collected garbage bin data to the Raspberry Pi Model B+.

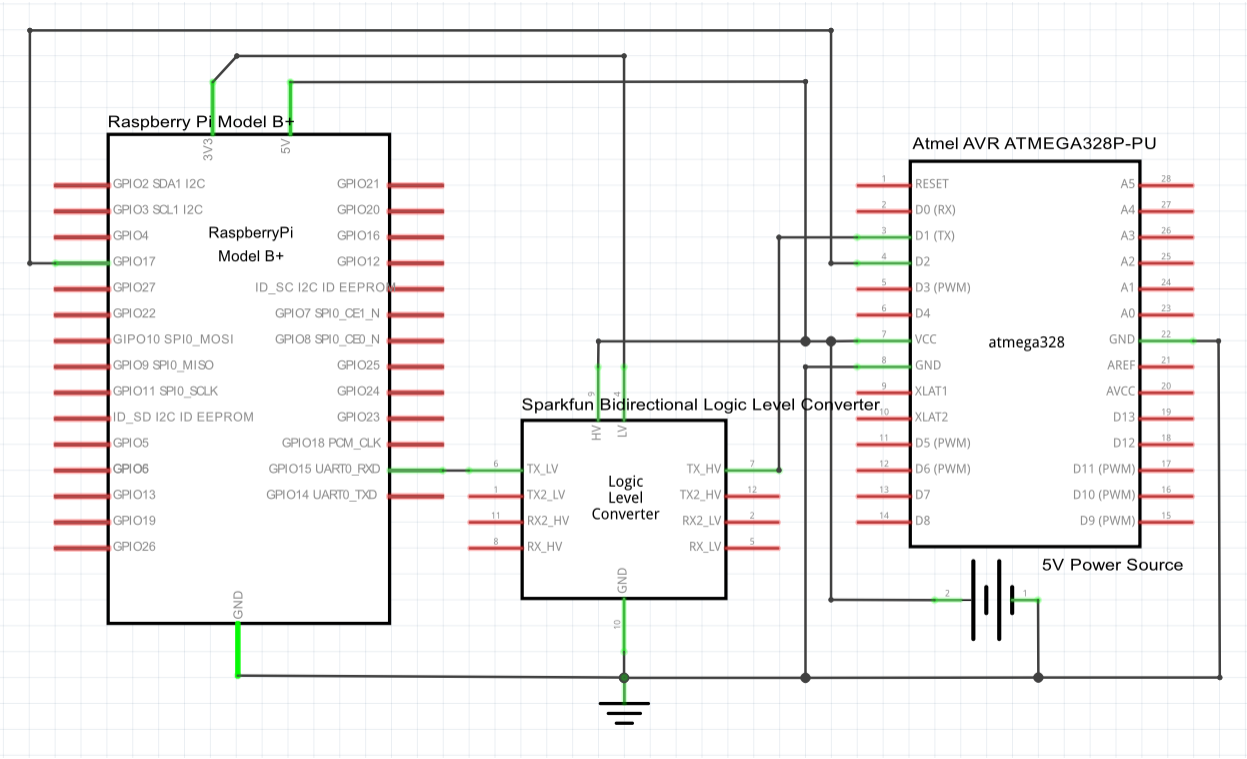


Figure 11: Schematic of the Connections Between the Raspberry Pi and the ATMEGA328P-PU

3.3.2.3 Hardware Drivers

As discussed in the previous subsection, the embedded software facilitates the collection of garbage bin data (e.g., ambient temperature, container depth, and location) by implementing several hardware drivers, each of which is responsible for operating a particular component in the integrated hardware. This subsection will discuss the design and implementation of the hardware drivers for the TMP36GZ temperature sensor, the HC-SR04 ultrasonic ranging module, and the Adafruit Ultimate GPS Breakout v3 module, as these components are critical to the collection of the aforementioned garbage bin data.

The first hardware driver to be discussed is the hardware driver for the TMP36GZ temperature sensor. In order to obtain a human-readable temperature value given a reading from the TMP36GZ temperature sensor, the following characteristic of the TMP36GZ temperature sensor is considered. In particular, the data sheet for the TMP36GZ temperature sensor states that the output voltage of the V_{out} pin on the TMP36GZ temperature sensor increases linearly by 10mV for each degree Celsius increase in the temperature [12]. Given this characteristic, the following sequence is executed by the hardware driver. First, the hardware driver obtains an ADC reading from the TMP36GZ temperature sensor via pin PC0 (i.e., pin 23) on the Atmel AVR ATMEGA328P-PU MCU [13]. Since the built-in ADC on the Atmel AVR ATMEGA328P-PU MCU has 10 bits of precision, this ADC reading will be represented as a discrete

quantization level Q_{ADC} from 0 to 1023[4] [13]. Second, the TMP36GZ temperature sensor takes this quantization level and maps it to a continuous voltage value using the formula:

$$V = \frac{Q_{ADC}}{2^b} \times V_{ref} \tag{8}$$

where $b = 10$ is the bit precision and $V_{ref} = 5000mV$ is the reference voltage [13]. Finally, the aforementioned characteristic is applied to convert this voltage representation V of temperature from millivolts to degrees Celsius by passing it into the following formula:

$$T = \frac{V - 500mV}{10mV/°C} \tag{9}$$

where 500mV is the output voltage yielded by the TMP36GZ temperature sensor at 0°C [12].

The second hardware driver to be discussed is the hardware driver for the HC-SR04 ultrasonic ranging module. In order to measure depth, the hardware driver first triggers the HC-SR04 ultrasonic ranging module by driving HIGH the TRIG pin on the HC-SR04 ultrasonic ranging module for a period of 10µs [11]. This has the effect of generating a 10µs pulse on TRIG, which the HC-SR04 ultrasonic ranging module interprets as a request [11]. As a result, the HC-SR04 ultrasonic ranging module will emit "an 8-cycle burst of ultrasound at 40kHz" [11]. It will also drive its ECHO pin HIGH until its receiver picks up the echoed ultrasound [11]. At this point, it should be intuitive that the round-trip-time RTT of the ultrasound, as represented by the width of the ECHO pulse from rising-edge to falling-edge, can be used to determine depth [11]. To do so, the hardware driver first computes the speed of sound v_{sound} using the ambient temperature T computed using the TMP36GZ driver and the following formula:

$$v_{sound} = 331m/s + (0.6m/s \cdot T) \tag{10}$$

The hardware driver then takes v_{sound} and the round-trip-time RTT in microseconds, and computes the depth in centimeters using the following formula:

$$\Delta d = \frac{v_{sound} \cdot \frac{RTT}{2}}{10{,}000} \tag{11}$$

In order to measure the round-trip-time of the emitted ultrasound, the hardware driver for the HC-SR04 ultrasonic ranging module makes use of the following resources onboard the Atmel AVR ATMEGA328P-PU MCU: TIMER1[5] and TIMER1 input capture interrupt[6] [13]. These resources are utilized as follows by the hardware driver. In the sequence of measuring depth, the hardware driver will initially have the TIMER1 input capture interrupt set to trigger whenever a rising-edge is detected on pin PB0 (i.e., pin 14) on the Atmel AVR ATMEGA328P-PU MCU [13]. Given this characteristic, the input capture interrupt will first be triggered when the HC-SR04 ultrasonic ranging module drives its ECHO pin HIGH [13]. At this point,

[4] For b bits of precision, the ADC reading would be represented as a quantization level from 0 to $2^b - 1$.
[5] TIMER1 is a 16-bit hardware timer onboard the Atmel AVR ATMEGA328P-PU MCU [13].
[6] Each time the TIMER1 input capture interrupt is triggered, the timestamp as reported by the TIMER1 counter, TCNT1, is stored in register ICR1 onboard the Atmel AVR ATMEGA328P-PU MCU [13].

a context switch occurs and control is transferred from the main program to the TIMER1 input capture ISR [13]. Within this ISR, the TIMER1 timer counter, TCNT1, is reset to 0 and the TIMER1 input capture interrupt is reset to trigger on a falling-edge [13]. A second context switch then occurs to return control to the main program [13]. After a short period of time, a falling-edge occurs on pin PB0 (i.e., pin 14) when the HC-SR04 ultrasonic ranging module drives its ECHO pin LOW after detecting an echoed ultrasound [13]. At this point, a third context switch occurs and control is once again transferred from the main program to the TIMER1 input capture ISR [13]. Given how the TIMER1 timer counter, TCNT1, was previously reset to 0, the timestamp from TIMER1 counter, TCNT1, stored in register ICR1 should now be equivalent to the round-trip-time of the emitted ultrasound in ticks[7] [13].

The final hardware driver to be discussed is the hardware driver for the Adafruit Ultimate GPS Breakout v3 module. In the case of the Adafruit Ultimate GPS Breakout v3 module, it will continuously stream serial GPS data in the form of National Marine Electronics Association (NMEA) sentences to the Atmel AVR ATMEGA328P-PU MCU via UART [14]. Given this characteristic, the hardware driver executes the following sequence in order to obtain the location of a garbage bin in terms of latitude and longitude. First, the hardware driver scans the serial data received via UART for the first global positioning system fix data (GPGGA) sentence [14]. Upon receiving the GPGGA sentence, the hardware driver then parses the GPGGA sentence for the current latitude and longitude. Here, the latitude and longitude is parsed from a GPGGA sentence by recognizing that a GPGGA sentence follows the following format:

```
$GPGGA,hhmmss.ss,llll.ll,a,yyyyy.yy,a,x,xx,x.x,x.x,M,x.x,M,x.x,xxxx
```

where `llll.ll` is a placeholder for the latitude and `yyyyy.yy` is a placeholder for the longitude [14].

3.3.3 Design Iterations

The design of the garbage bin sensors subsystem has undergone several small design iterations before converging to the design discussed in the previous subsection. In the beginning, the design of the garbage bin sensors subsystem was centered on the Arduino UNO rather than directly on the Atmel AVR ATMEGA328P-PU MCU; however, in the interest of reducing the size dimensions and the monetary cost of the integrated hardware for this subsystem, the entire design of the garbage bin sensors subsystem was transitioned over to the Atmel AVR ATMEGA328P-PU MCU.

From there, the following additional small design iterations took place. First, a 16MHz external crystal oscillator was added to the integrated hardware, as the internal 8MHz RC oscillator present on the Atmel AVR ATMEGA328P-PU was deemed insufficient for performing the time-sensitive measurements required for measuring container depth. This was largely because the clock rate of this internal 8MHz RC oscillator was found to deviate with temperature and voltage [13]. Second, the TMP36GZ temperature sensor was also added to the integrated hardware so that the ambient temperature of the garbage bin could be obtained from the environment rather than as a hardcoded constant. Third, the master/slave intercommunication model between the integrated hardware (as the slave) and the Raspberry Pi Model

[7] TIMER1 is defined with a prescaler of 8 to reduce its clock rate by a factor of 8 from 16MHz (i.e., the clock rate of the Atmel AVR ATMEGA328P-PU MCU) to 2MHz, so each tick accounts for 0.5µs in time [13].

B+ (as the master) was transitioned from a synchronous approach to an asynchronous approach. In particular, it used to be the case that the integrated hardware would collect garbage bin data on a periodic basis and send the most recently collected garbage bin data to the Raspberry Pi Model B+ when prompted. The problem that arose with this synchronous approach was that the Raspberry Pi Model B+ could potentially interrupt the integrated hardware in the middle of garbage bin data collection. Consequently, the asynchronous approach was adopted, where the integrated hardware only performs garbage bin data collection on request from the Raspberry Pi Model B+.

3.3.4 Design Alternatives

In addition to the aforementioned small design iterations, the following design alternative was initially considered over the current implementation for the asynchronous master/slave model discussed in section 3.3.2.2. In this design alternative, the serial peripheral interface (SPI) on the Atmel AVR ATMEGA328P-PU is used instead of the UART for data transmission to the Raspberry Pi Model B+. Furthermore, the external hardware interrupt that is present in the implementation of the asynchronous master/slave model in section 3.3.2.2 is absent from this design alternative.

The primary reason why this SPI-based design alternative was initially considered over the currently implemented UART-based design had to do with the advantages that SPI had over UART. In particular, the throughput rate for SPI is much higher than the throughput rate for UART. Using SPI, data could be sent to the Raspberry Pi Model B+ at a bit rate of 864,600 bits per second [13]. In contrast, data transmission using UART was limited to 9,600 bits per second [13]. Additionally, the use of SPI over UART would enable the possibility of connecting the Raspberry Pi Model B+ to two slaves instead of just one[8] [13]. It would also free up the hardware UART transmit and receive pins on both the Atmel AVR ATMEGA328P-PU MCU and the Raspberry Pi Model B+ for other applications, which made this design alternative very attractive.

Unfortunately, there were also several disadvantages tied to using SPI over UART. In particular, the SPI-based design required the use of 4 hardware pins on both the Atmel AVR ATMEGA328P-PU and the Raspberry Pi Model B+ [13]. Additionally, the SPI on the Atmel AVR ATMEGA328P-PU MCU requires each byte of data to be sent one at a time [13]. In contrast, the UART on the Atmel AVR ATMEGA328P-PU allows for an entire buffer of data to be sent at a time [13]. As a result, an additional protocol had to be defined on top of the SPI driver for the Atmel AVR ATMEGA328P-PU in order to synchronize the transmission of garbage bin data between the integrated hardware and the Raspberry Pi Model B+.

The synchronization protocol works as follows. To begin, the Raspberry Pi Model B+ requests garbage bin data collection by sending a start of transmission byte, 0x01, via the SPI to the integrated hardware. The integrated hardware, upon receiving the start of transmission byte, 0x01, would then send a start of text byte, 0x02, via the SPI to the Raspberry Pi Model B+. It would then proceed to collect the necessary garbage bin data and place it into the data buffer. In the meantime, the Raspberry Pi Model B+, which has just received the start of text byte, 0x02, sends an acknowledgement byte 0x06 to the Atmel AVR ATMEGA328P-PU MCU. The Atmel AVR ATMEGA328P-PU MCU, upon receiving this acknowledgement

[8] Both the Atmel AVR ATMEGA328P-PU MCU and the Raspberry Pi Model B+ have only one set of hardware UART transmit and receive pins [13].

byte 0x06, will then send the next byte in the data buffer to the Raspberry Pi Model B+. In response, the Raspberry Pi Model B+ sends another acknowledgement byte, 0x06, to the Atmel AVR ATMEGA328P-PU MCU. This cycle is repeated until all bytes in the data buffer have been sent to the Raspberry Pi Model B+, in which case the Atmel AVR ATMEGA328P-PU MCU will then send an end of text byte, 0x03, to the Raspberry Pi Model B+. The Raspberry Pi Model B+ receives the end of text byte, 0x03, and sends the end of transmission byte, 0x04, to the ATMEGA328P-PU MCU to indicate that the transmission is over.

3.3.5 Subsystem Data and Analysis

3.3.5.1 Simulation and Critical Evaluation

In order to select the final design for the asynchronous master/slave model used in the garbage bin sensors subsystem, the UART-based and SPI-based design alternatives were systematically evaluated against the weighted decision matrix in Table 6 below.

Table 14: The Weighted Decision Matrix for the Garbage Bin Sensors Subsystem Design Alternatives

Criteria	Weight	Points	Requirements
Percent Program Memory Usage	25%	See req.	$Points = \frac{100 - \%Program\ Memory\ Usage}{100} \times 3$
Percent Data Memory Usage	25%	See req.	$Points = \frac{100 - \%Data\ Memory\ Usage}{100} \times 3$
Quantity of Pins Required	10%	0	Requires 7 or more pins.
		1	Requires 5 to 6 pins.
		2	Requires 3 to 4 pins.
		3	Requires 0 to 2 pins.
Reliability	40%	0	Data transmission fails or exhibits undefined behavior.
		1	Data transmission succeeds at least 1/3 of the time.
		2	Data transmission succeeds 2/3 of the time.
		3	Data transmission always succeeds.

Table 6 evaluates the design alternatives for the garbage bin sensors subsystem against the following criteria: percent program memory usage, percent data memory usage, quantity of pins required, and robustness. The percent program memory usage evaluates the percentage of the 32kB flash memory on the Atmel AVR ATMEGA328P-PU MCU taken up by the embedded program implementing the design alternative. The percent data memory usage evaluates the percentage of the 32kB flash memory on the Atmel AVR ATMEGA328P-PU MCU taken up by memory allocated on the stack by the embedded program implementing the design alternative. The quantity of pins required evaluates the number of pins required by the design alternative. Finally, reliability evaluates how often garbage bin data transmission succeeds under a given design alternative.

In general, this analysis emphasizes the simplicity and robustness of the design. As a result, the weightings for each criterion were assigned as follows. The reliability criterion was assigned a weighting of 40% because first and foremost, the implementation yielded from the design alternative must be reliable. Meanwhile, the percent program memory and percent data memory criterion were each assigned weightings of 25% (the second highest weighting) because memory usage is arguably the second most important criterion due to the limited amount of flash memory available on the Atmel AVR ATMEGA328P-

PU MCU. In particular, a lower memory footprint is strongly preferred because the Atmel AVR ATMEGA328P-PU MCU has been known to exhibit undefined behavior when memory limits are exceeded [13]. Finally, the quantity of pins criterion is assigned the lowest weighting of 10% because it is considered the least important. While design alternatives using fewer pins are desirable given the limited number of hardware pins on the AVR ATMEGA328P-PU MCU, robustness is more important.

To evaluate the design alternatives for the garbage bin sensors subsystem, the hardware and software component of each design alternative was prototyped. For the software component, the embedded software was prototyped in AVR embedded C using the Atmel Studio 6 integrated development environment (IDE). Afterwards, each design alternative was evaluated against each criterion in Figure 6 as follows. For the percent program memory and percent data memory criteria, the values for percent program memory and percent data memory were obtained from the compile logs for the embedded programs pertaining to each design alternative. For the quantity of pins required criterion, the number of non-power and non-ground pins on the AVR ATMEGA328P-PU MCU used by the design alternative for data transmission is counted. Finally, for the reliability criterion, the data transmission on the garbage bin sensors subsystem is tested 300 times and the fraction of successful data transmissions is taken. The evaluation results for each design alternative against Table 6 are provide in Tables X and Y.

Table 15: Evaluation Results for UART-based Implementation of Asynchronous Master/Slave Model

Criteria	**Weight**	**Result**	**Points**	**Weighted Score**
Percent Program Memory Usage	25%	% Program Memory Usage = 23.9%	2.283	0.57075
Percent Data Memory Usage	25%	% Data Memory Usage = 11.2%	2.664	0.666
Quantity of Pins Required	10%	Quantity of Pins Required = 2	3	0.3
Reliability	40%	Success Rate = 300/300	3	1.2
Total				2.73675

Table 16: Evaluation Results for SPI-based Implementation of Asynchronous Master/Slave Model

Criteria	**Weight**	**Result**	**Points**	**Weighted Score**
Percent Program Memory Usage	25%	% Program Memory Usage = 18.2%	2.454	0.6135
Percent Data Memory Usage	25%	% Data Memory Usage = 4.9%	2.853	0.71325
Quantity of Pins Required	10%	Quantity of Pins Required = 4	2	0.2
Reliability	40%	Success Rate = 1/300	0	0
Total				1.52675

Based on the results provided in Tables 7 and 8, the UART-based asynchronous master/slave model implementation yielded the highest weighted score; therefore, it was selected for implementation in the garbage bin sensors subsystem.

3.3.5.2 Satisfaction of Project Specifications

For ease of reading, the project specifications from Section 2 that are relevant to the garbage bin sensors subsystem are reproduced below:

1. There must be a latency of less than 5 seconds when measuring garbage volume.
2. The garbage bin sensors subsystem must be able to measure and report depth measurements that are within ±10% accuracy of a manually produced measurement.

In order to demonstrate that the two project specifications above are satisfied by the garbage bin sensors subsystem, the following tests were carried out with respect to each project specification. To prove that specification no. 1 was satisfied, the garbage bin sensors subsystem was used to measure its distance from a wall distanced 200cm away in order to simulate container depth measurement in a garbage bin 2m deep. Here, it is noted that the latency of the container depth measurement is dominated by the round-trip-time RTT of the emitted ultrasound. As a result, a modified embedded program for the garbage bin sensors subsystem that additionally returns RTT was used to approximate the latency when measuring garbage volume in an extreme case. To prove that specification no. 2 was satisfied, the garbage bin sensors subsystem was positioned 8cm away from a solid object and prompted to perform a depth measurement. The percentage error between the measured depth and the expected depth of 8cm was then taken. Each one of these tests was carried out a total of 3 times and the results are shown in Tables 17 and 18 below.

Table 17: Garbage Bin Sensors Subsystem Data for Specification No. 1

Trial	Expected Depth (cm)	Latency (s)	Specification No. 1 Satisfied
1	200	0.023761	Yes
2	200	0.023755	Yes
3	200	0.023748	Yes

Table 18: Garbage Bin Sensors Subsystem Data for Specification No. 2

Trial	Expected Depth (cm)	Measured Depth (cm)	%Error	Specification No. 2 Satisfied
1	8	7.644778	4.440%	Yes
2	8	7.535055	5.812%	Yes
3	8	7.483455	6.457%	Yes

As shown in Tables 17 and 18, specifications no. 1 and no. 2 appear to be satisfied in general by the garbage bin sensors subsystem. In particular, the latency, even when measuring a container of depth 200cm, should not exceed 5s. From the results shown in Table 17, the measured latencies do not even exceed 1s. Similarly, the results in Table 18 show that the measured depth closely follows the expected depth of 8cm given that percentage error is less than 10% in each case.

4. Discussion and Conclusions

4.1 Evaluation of Final Design

Overall, the final design of the Distributed Garbage Collection System follows relatively closely to the project objectives and design specifications with a few minor exceptions. Specifically, the biggest being the removal of the autonomy of service of the final product design. Further, a number of additional specifications had been removed as a result of design tweaks since initial conception. These include the removal of specifications such as minor design features (e.g., disconnection indicator) and processes related to the former design, which was intended to contain an autonomous subsystem.

The removal of the autonomous subsystem and related functional specifications came due to constraints based on knowledge and time. It was decided that given the time remaining in the design process, acquiring the knowledge required to integrate all other subsystems with an automatically traversing distributed network of nodes would be far too costly in time. To elaborate, upon research into this system, group members found that they did not have the knowledge to provide these robotics the ability of turning and other features seen vital to autonomy. Also, members lacked a deep enough knowledge of circuitry to develop controllers for such a control system. Ultimately, it was suggested via consultation that this subsystem be ignored going forward.

Further, specifications such as the disconnection indicator and idle state bins became non-essential specifications as they were seen as not vital to the project's final design. As time constraints became a pressing issue in the ongoing design, these features which were once essential are now more "nice to have" and were not investigated further. The remaining essential features found in Section 2.1, however, were met and implemented as core features in the final design. Finally, in terms of non-functional specifications, all expected specifications were met with the exception of the 6 hour battery specification. This once essential feature has since become non-essential as it is seen as too trivial given the removal of the autonomous nature of the original design.

4.2 Use of Advanced Knowledge

Contained within the design of the Distributed Garbage Collection System are the contents of several upper-year University of Waterloo ECE courses. In general, this project's design relies heavily on many software-based concepts, with the addition of a number of hardware, and computer networking concepts. In particular, ECE 455, ECE457A and ECE 250 play a vital role in the software subsystems of the design and the software sitting on top of the hardware design, which courses such as ECE 416 and ECE 358 are considered very important to the design of the networks component. Finally, concepts learned in ECE 222 and ECE 224 form the foundation for the understanding of design's hardware aspects.

To elaborate, subject matter from ECE 457A, such as k-means clustering and trajectory-based search algorithms, can be found in the software of the garbage bin allocation subsystem. Additionally, ECE 250 provides the overall basis for the understanding of algorithm design. A basic understanding from ECE 250 allows for the design of efficient, fast runtime and low space algorithms for all subsystems containing software. Further, the software concepts learned in ECE 455 are the basis for programming timers and interrupts in the garbage bin sensors subsystem. In addition to the interrupts concepts learned in ECE 455, the hardware foundations learned in ECE 222 and ECE 224 provide insight on reliable hardware and

interrupt usage. Finally, the use of computer networking theory from ECE 358 and ECE 416 are vital to the design of the network communications subsystem. The understanding of TCP and UDP are the foundation of the design used to transfer messages from the server to the client garbage bins. This allows for bin capacities to be known remotely and enables for this information to be displayed via the user interface subsystem. Moreover, the use of probability theory through the application of the Poisson distribution aids in determining the overflow probability of the many garbage bins which may exist in the network.

4.3 Creativity, Novelty, Elegance

From a novelty perspective, while smart garbage bins do exist in practice, it is rare to find a system of garbage bins that can be dynamically allocated based on usage statistics. The novelty of this design lies in the fact that the system is able to gradually master garbage bin allocation by simply collecting and processing garbage bin data over time. The allocation of garbage bins is updated as time progresses using an allocation policy to effectively adapt to changes in usage patterns throughout the day. Additionally, the design is intended to bring humanity one step closer towards completely automating the seemingly mundane task of garbage collection, especially if autonomous movement and navigation of the garbage bin becomes a feasible target in the near future.

4.4 Quality of Risk Assessment

In the initial design of the Distributed Garbage Collection System, the most obvious potential risk was the battery sensor subsystem. To mitigate this risk in the final design, it was ultimately decided that it should be removed. The reason for the removal of this subsystem and therefore, the risk associated with it, was due to the removal of the autonomy in the overall project. Because the autonomy of the project no longer exists, there is therefore no longer a need to a battery, which was ultimately to be used for mobility. On the other hand, however, another risk which arose in the final design is the nature of the final circuit of the sensors subsystem. That is, the circuit as it remains presently is unenclosed and left open to the environment. This can cause issues where the surroundings may potentially create a short in the circuit, create shocks for users or even electrical fire. Though the current running through this circuit is likely not enough to cause harm, it is expected that an enclosure should be developed for the circuit which would protect it from any of the associated risks mentioned.

4.5 Student Workload

Given the final design of the Distributed Garbage Collection System, the workload provided by members of the design team can be seen to be relatively evenly distributed with a few exceptions. Specifically, about 35% of total labor hours included in the overall tasks of this project was taken upon by Justin Cheung, and 20% by Eugene Nam, while the remaining work was split relatively evenly between the remaining three group members. The reason for this one exception is likely due to the amount of research work and development done by Justin and Eugene during the 2014 work term. Additionally, while the remaining group members contributed to the many subsystems, Justin Cheung was the sole individual working on the subsystem considered the most creative and novel in the final design.

References

[1] F. Council, "Litter facts: why do people litter?," [Online]. Available: http://www.falkirk.gov.uk/services/corporate_neighbourhood/estates_management/litterzone/litter_strategy/litter_facts/why_do_people_litter.aspx. [Accessed 24 May 2014].

[2] L. I. W. TOO!, "Why do people litter?," [Online]. Available: http://www.litteringiswrongtoo.org/faq#people. [Accessed 24 May 2014].

[3] Movable Type Ltd., "Movable Type Scripts: Calculate distance, bearing and more between Latitude/Longitude points," Movable Type Ltd., [Online]. Available: http://www.movable-type.co.uk/scripts/latlong.html. [Accessed 1 February 2015].

[4] D. Arthur and S. Vassilvitskii, "k-means++: the advantages of careful seeding," in *SODA '07 Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, Philadelphia, 2007.

[5] M. Badreldin, "A Metaheuristic Approach to Multi-Robot Task Allocation," Cairo, 2013.

[6] M. Badreldin, A. Hussein and A. Khamis, "A Comparative Study between Optimization and Market-Based Approaches to Multi-Robot Task Allocation," Hindawi Publishing Corporation, Suez, 2013.

[7] Wikipedia contributors, "Hypertext Transfer Protocol," Wikipedia, The Free Encyclopedia., 04 02 2015. [Online]. Available: http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol. [Accessed 10 02 2015].

[8] Wikipedia contributers, "XML," Wikipedia. The Free Encyclopedia, 31 01 2015. [Online]. Available: http://en.wikipedia.org/w/index.php?title=XML&oldid=645028719. [Accessed 11 02 2015].

[9] D. Crockford, "JSON," [Online]. Available: http://json.org/. [Accessed 10 02 2015].

[10] J. Nielsen, "Website Response Times," June 2010. [Online]. Available: http://www.nngroup.com/articles/website-response-times/. [Accessed 27 December 2014].

[11] Elec Freaks, "HC-SR04 Ultrasonic Module User Guide," 12 August 2013. [Online]. Available: http://www.elecfreaks.com/5464.html. [Accessed 27 January 2015].

[12] Analog Devices, "Analog Devices," 2013. [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/TMP35_36_37.pdf. [Accessed 29 January 2015].

[13] Atmel, "Atmel," 2012. [Online]. Available: http://www.atmel.com/Images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet.pdf. [Accessed 19 January 2015].

[14] G. Baddeley, "GPS - NMEA sentence information," 20 July 2001. [Online]. Available: http://aprs.gids.nl/nmea/. [Accessed 14 January 2015].

Appendix A: Completed Prototype Hazard Disclosure Form

ECE498B: Prototype Hazard Disclosure Form* Group number: 2015.034

Instructions: Answer all the following questions by putting an *X* in either the yes or no column. If unsure, answer yes. If you answer yes to any question, set up an appointment with the Lab Instructor to ensure your prototype is safe for the symposium. Include this completed form in your Final Report (Appendix A) even if you answer no to all questions.

Question: Does your prototype…	yes	no
1. include any circuitry that you designed or built by yourself?	X	
2. include any circuitry that is not enclosed in an approved plastic or metal electrical box? (Note: Likely will be in box by Symposium date.)	X	
3. involve any 120V AC circuitry/device that is not approved by CSA or ESA?		X
4. involve circuitry that is not connected to its power supply with a fuse and a switch? (Note: Will hopefully have a fuse by Symposium date.)	X	
5. use high-capacity or high-density (e.g., lithium-ion) batteries?		X
6. emit non-trivial amounts of RF radiation?		X
7. involve x-rays or radioactive materials?		X
8. use unprotected lasers of any class?		X
9. involve strobe lights?		X
10. have exposed moving parts that may pinch, hit, or crush a person?		X
11. involve projectiles or any part that can fly?		X
12. have exposed sharp edges or points?		X
13. use high-pressure gases or liquids?		X
14. involve irritating or dangerous chemicals (assume all nano-materials are dangerous)?		X
15. involve any biological materials (dead or alive) or any food/drink?		X
16. emit dangerously loud sounds?		X
17. eject gas, particles, or fluids into the environment?		X
18. involve accessible components that reach temperatures above 40°C or below 0°C?		X
19. involve any other hazard? Describe:		

Lab Instructor Inspection Report (needed only if "yes" is checked to any question above)

☐ Prototype is deemed to be unsafe in its current state and requires another inspection after changes are made.

☐ Prototype is deemed to be conditionally acceptable† for the symposium, and another inspection is not required.

☐ Prototype is deemed to be safe for the symposium in its current state.

†State on the back of this form what minor changes are required for the prototype to be considered safe for the symposium.

Lab Instructor Signature _____ Date _____

Project Consultant Inspection Report at Final Prototype Demonstration (needed for all projects)

yes	no	
☐	☐	The group appears to have accurately answered the above 19 questions.
☐	☐	The prototype presently appears to be safe for the public symposium.

Consultant Signature _____ Date _____

Appendix B: Completed Symposium Floor Plan Request Form

ECE498B: Symposium Floor Plan Request Form* Group number: 2015.034

Instructions: Answer all the following questions by putting an *X* in either the yes or no column. Provide additional information in the far right column as requested. Include the completed form in Appendix B of your Final Report, even if you answer no to all questions. Requests for equipment (e.g., an oscilloscope, a power supply, a large monitor, a computer, or a lab stool) should not be included on this form; instead, for such equipment requests use the on-line reservation system as described in the *Symposium Checklist and Schedule* document.

Question:	yes	no	if you answered "yes"...
1. Does your project require one or more hardwire internet connections at the symposium? **Note:** We provide Ethernet connections only, via a male RJ-45 connector. The connection comprises a static IP address in the uwaterloo.ca domain on a shared 100Mbps link. Do not count on the DC building wireless system being available for your project.	X		If yes, state how many connections you require: 2 internet connections
2. Do you desire to use your own wireless router at the symposium? If yes, you will need to have your setup approved well before the symposium date. Unapproved routers are strictly prohibited.		X	If yes, contact Paul Ludwig for details as soon as possible.
3. Each booth has a 7.5-amp 6-outlet power bar. Does your project require more than 7.5 amps of mains (120 V AC) power?		X	If yes, state how many amps you require in total:
4. Each booth has a 7.5-amp 6-outlet power bar. Does your project require more than 6 outlets?		X	If yes, we will supply your booth with two power bars.
5. Do you intend to put any electronics/computers on the floor under your booth?		X	If yes, we will ensure you can do this safely.
6. Does your project involve projectiles or flying parts? (We have a large "cage" at the symposium for projects that involve projectiles or flying parts.)		X	If yes, state the nature of the projectile or flying part:
7. Does your project require special lighting conditions (e.g., dim light, bright light)? We will do our best to accommodate lighting requests, but no guarantees are made.		X	If yes, state the nature of the desired lighting conditions:
8. The default booth consists of a 5' wide by 2.5' deep table. The table is 2.54' tall. Does your project require extra table space (giving you a total table area of 7.5' wide by 2.5' deep)? We will do our best to accommodate space requests, subject to the urgency of the request and overall space and safety constraints.	X		If yes, explain specifically why you are requesting the extra table space: We want an external monitor connected to a laptop.
9. The default floor space, including the table and area for you/visitors to stand, is around 6' by 6'. Does your project require extra floor space? We will do our best to accommodate space requests, subject to the urgency of the request and overall space and safety constraints. **Note:** Due to the extra-large class size this year, very few requests for extra floor space will be granted.		X	If yes, explain why you need the extra floor space: State how much extra space you are requesting:
10. Do you have any other special floor plan requests?	X		State your request: External Monitor. Ideally 27 inches.