BinBot
Design Document I

REVISION HISTORY

Revision #	Author	Revision Date	Comments
1.0	Michael Savitski	October 5, 2019	Outline
1.1	Michael Savitski	October 6, 2019	Machine Learning
1.2	Sean DiGirolamo	October 6, 2019	Server Communication
1.3	Jose Silva	October 6, 2019	Robot Kit
1.4	Kwamina Thompson	October 6, 2019	Mobile Application
1.5	Sean Reddington	October 6, 2019	System Overview, Robot Kit additions, formatting
2.0	Sean DiGirolamo	October 13, 2019	Added Server architecture
2.1	Sean Reddington	October 19, 2019	Added full system overview, server architecture description, mobile app class diagram and architecture description, sequence diagram and description, formatting and clean up for resubmission

Table of Contents

System Overview

BinBot is a waste-collection robot intended to patrol specific areas such as university grounds, stadiums, boardwalks, or schools. BinBot will identify waste that is laying on the ground and collect it to be disposed of properly. BinBot will have a camera module and on-board microcontroller unit that communicates with a server via wi-fi. To outsource the heavy data processing from the on-site robot; a Linux server will process the images sent by BinBot using data representations created with a deep learning algorithm to identify pieces of waste. The server will then inform BinBot of information regarding the photos, such as if any waste has been identified, and if so, how far it is and how BinBot should navigate to the waste. The waste will then be collected using a mechanical arm and place it in a waste bin attached to the robot. The image feed from BinBot, with additional visual indicators of identified waste, can be transmitted to a mobile application for observing BinBot's progress and success.

One of the key components for BinBot to function as needed will be deep learning software for training a neural network model, so that BinBot's other software will be able to identify waste objects in the images from its camera module. This will entail the use of the OpenCV library for processing images, and the use of the open-source deep learning library TensorFlow. Ideally, this separate software component will be run on a GPU supported computer system. The neural network data model will be trained using hundreds to thousands of images of waste objects, captured in the expected resolution of BinBot's camera. Also included in the training data with these images will be information such as camera height, object size, and object distance, so that BinBot will be able to make estimations to allow it to traverse to the waste objects and successfully collect them. After feeding images to the neural network, the neural network will be tasked with returning whether or not waste is located in the image, and if so, how far the waste is and at what angle it is from the robot.

The completed data model will be applied to software which will run on a server along with a communication socket service, which BinBot will continuously connect to via wi-fi. The images continuously collected by BinBot's camera module will be sent to the server for processing via the OpenCV library, which will use the trained neural network data model to identify waste objects. Once the waste has been identified, as well as the distance and angle, movement instructions will then be calculated and sent back to BinBot's on-board computer board so that it may travel to the nearby waste objects and pick them up. For the purposes of this simple project, BinBot will simply be instructed to turn towards the waste, and travel to it in a straight line.

BinBot's on-board computer will be an Arduino Mega 2560 board with limited functionality. It will be to collect images from the camera module and then transmit them to the data processing API. Receiving data back from the data processing API about waste objects in BinBot's camera's view, the primary purpose of the board will be to operate BinBot's robotic components. BinBot will have robotic treads for traversing across the floor to approach waste objects, as well as a robotic arm for collecting the objects. The software running on the board will need to use the constantly updated information about distance and size of identified objects to operate these components efficiently and successfully collect waste.

Additionally, a companion mobile application will be developed for users of BinBot to install on Android devices. This companion application will be able to receive images from the data processing server which will allow the user to watch a live feed of BinBot's camera view. The server will modify these images visually to have boxes around identified waste objects, and text about BinBot's current progress in its process of collecting the objects such as the estimated distance and status messages like "traversing" or "collecting".

System Block Diagram

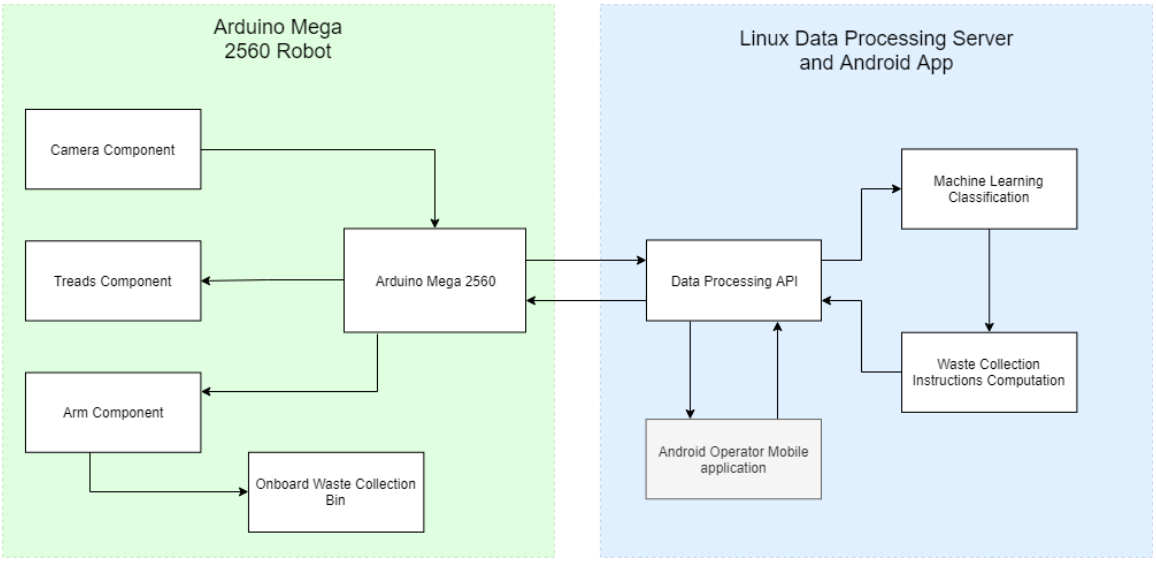

Fig 1. Block diagram illustrates the flow of data from the Camera feed to BinBot disposing of the waste.

The block diagram shows a high level overview of BinBot project's system architecture. Each of the hardware components on the robot kit will have a corresponding controller interface in the Sketch program loaded onto the Arduino Mega 2560. The Arduino Mega 2560 will exchange data with the Linux server over Wi-Fi via a data transmission interface. The Linux server will be hosting a data processing API that will pass BinBot's camera feed to the machine learning classification processing; passing the results to the Android operator mobile application. If there is collectable waste, the server will then computer instructions for the robot to collect the target waste object and send it back to the Arduino board. The Arduino will then pass the instructions to the tread and arm interfaces to collect the object, placing it into an onboard waste collection bin.

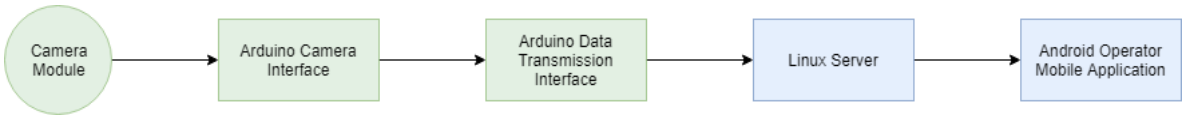

Fig 2. Diagram displaying interaction between BinBot's Camera Module, Linux Server, and Android Application.

The camera module will capture images at 1 fps via the Camera Interface. This image feed will be passed to the Arduino's Data Transmission interface which will send the images to the Linux server to be processed. Along with the data processing, the image feed will be passed to the Android operator mobile application for monitoring and manual intervention of BinBot during testing.

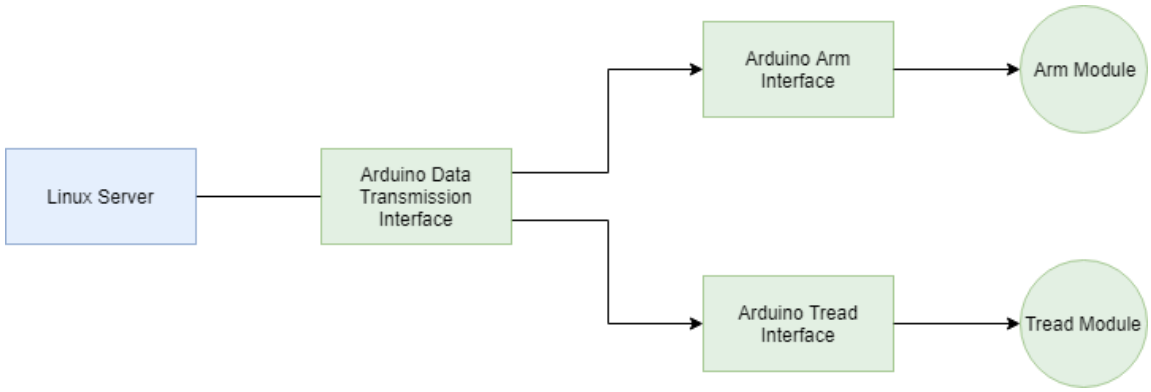

Fig 3. Diagram displaying interaction between the Linux Server and the Arduino's interfaces in order to operate the robot.

The Linux Server will exchange data with the Arduino over Wi-Fi through the Arduino's Data Transmission interface. This includes computed instructions for BinBot to travel to or collect the target object. The Arduino will then pass the instructions to the corresponding interface. These interfaces will provide the functions to move the arm or tread modules.

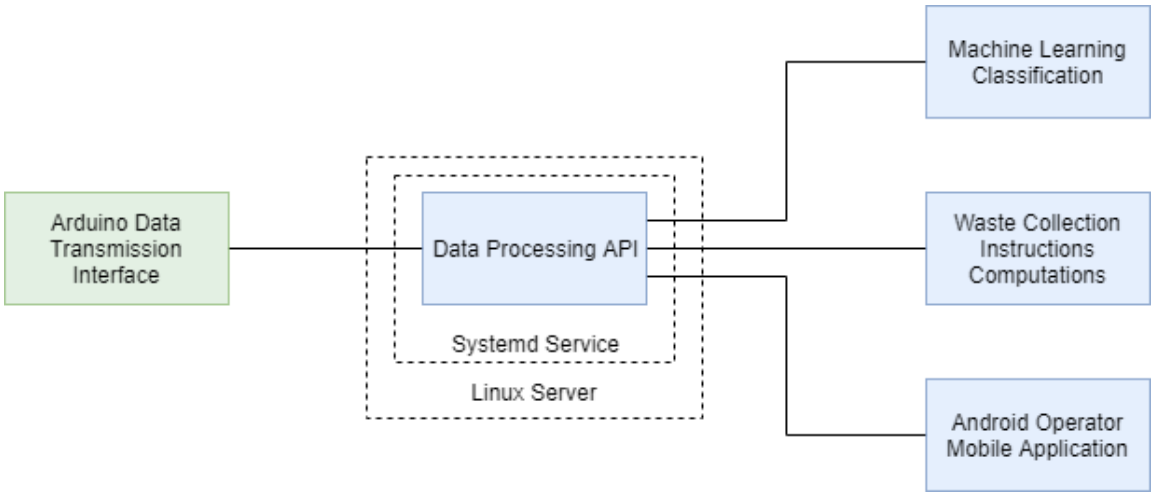

Fig 4. Diagram displaying the interaction between the Arduino's Data Transmission Interface, Linux data processing server, and Android Application.

The Arduino Data Transmission interface will send the image feed to the Linux processing server. The proposed method is to host a data processing API application that will execute via Systemd services. The data processing API will relay the image feed to the machine learning classification processing. If the machine learning model identifies collectable waste, instructions for the robot kit to collect the waste will be computed. These instructions include navigation instructions for the tread module and collection instructions for the arm module. The image including the results of the object classification will also be sent to the Android mobile application.

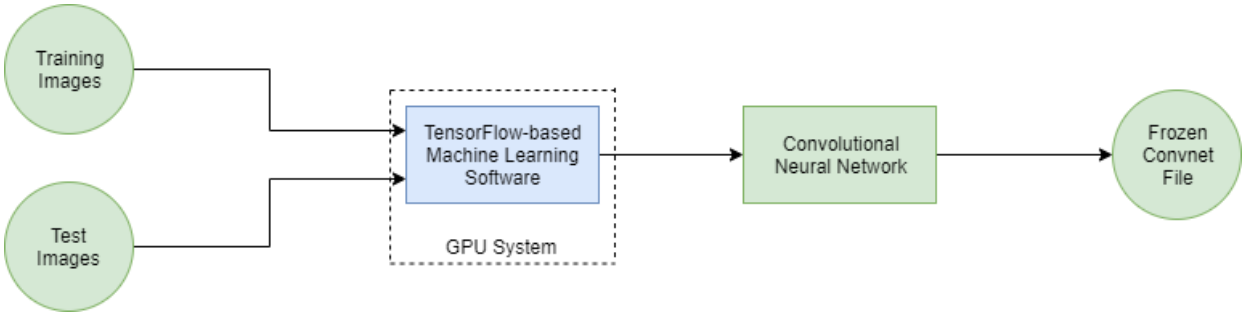

Fig 5. Diagram illustrating the process of training the neural network data model that will enable BinBot to recognize objects.

The machine learning algorithm will take as input approximately one thousand images for each trash objects we want to make BinBot's image recognition able to identify. Running on a GPU system and implementing Google's TensorFlow API, the software will train a convolutional neural network data model. There will also be several hundred images reserved as a test data set, to verify the neural network's success on data it was not trained on. This model can then be exported, a process called "freezing", to be implemented in the server-side OpenCV implementation.

ROBOT KIT

BinBot is a robotic tank car kit with a 4 DOF robotic arm, video camera and proximity modules, Wi-Fi and Bluetooth controllers, powered by an Arduino Mega 2560. The Arduino Mega 2560 will be loaded with an Arduino Sketch that includes a main class and four realized interfaces; *Transmission*, *Camera*, *Treads*, *Arm*.

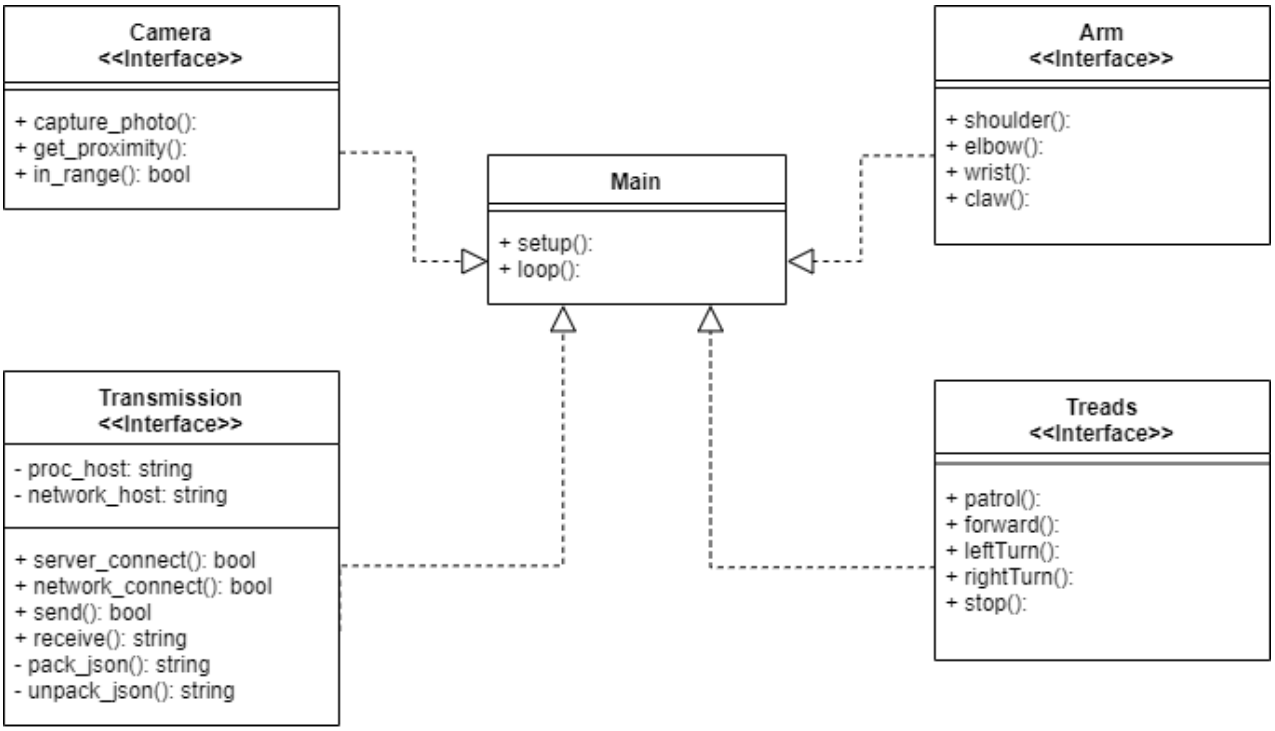

Figure 6: UML Class Diagram of classes and interfaces used to implement the mechanics of BinBot

The main class will consist of two functions; *setup()* and *loop()*. Once power is applied to the Arduino, the *setup()* function will be called. This function will be used to make any initial configurations prior to BinBot starting its automated waste collection process, such as establishing connections to the network Wi-Fi and Linux processing server.

The Transmission interface will be used to exchange data between the Arduino and the Linux server. The *network_connect()* will be used to establish a connection to the designated network, i.e., Temple University's secure wireless network. Next, the *server_connect()* function will be called in order to establish a connection between the Arduino Wi-Fi controller and the Linux processing server. The Arduino will then be ready to call the *send()* and *receive()* functions to exchange data with the server. The transmission interface will also include functions, *pack_json()* and *unpack_json()*, to handle JSON data structures.

BinBot has a second-generation HD camera with manual adjustment of focus. The camera has two degree of freedom and records at 30 frames per second. BinBot will use this camera to take a photo of the environment in front of it calling the *capture_photo()* method in the Camera interface. This method will be called at a fixed time interval that will allow BinBot to check if there is waste in its range via a Linux processing server.

If no waste is present in the photo BinBot will then call the patrol() method which will be in the Treads interface, allowing BinBot to move slightly in a new direction (right() or left() methods that moves BinBot in the specific direction) to continue searching for waste. If an item is present in the photo BinBot will call the forward() method, which calls the treads to move in a forward direction. BinBot will stop after a specific amount of time and once again call the *capture_photo()* method to check how far it is from the item and check if it able to classify it as waste. If the item is classified as waste BinBot will continue *forward()* until in proper range of the item which will done by using the in_range*()* method. Once in proper range the robotic arm will then called into use.

BinBot is designed with a 4 DOF robot arm with a claw like hand at the end of it. This will be the mechanism used to pick up the identified waste. Once in range the Arm class methods will be called to pick up the waste. These methods include *shoulder(), elbow(), wrist(),* and *claw()* which each help control an individual component of the robot arm.

PROCESSING SERVER

The data processing server will be a Linux server hosting a Java application including 5 main components: OpenCVWrapper, Instructions, BotConnection, and AppConnection. This diagram shows a high level UML diagram featuring the initial design of the processing server's implementation.

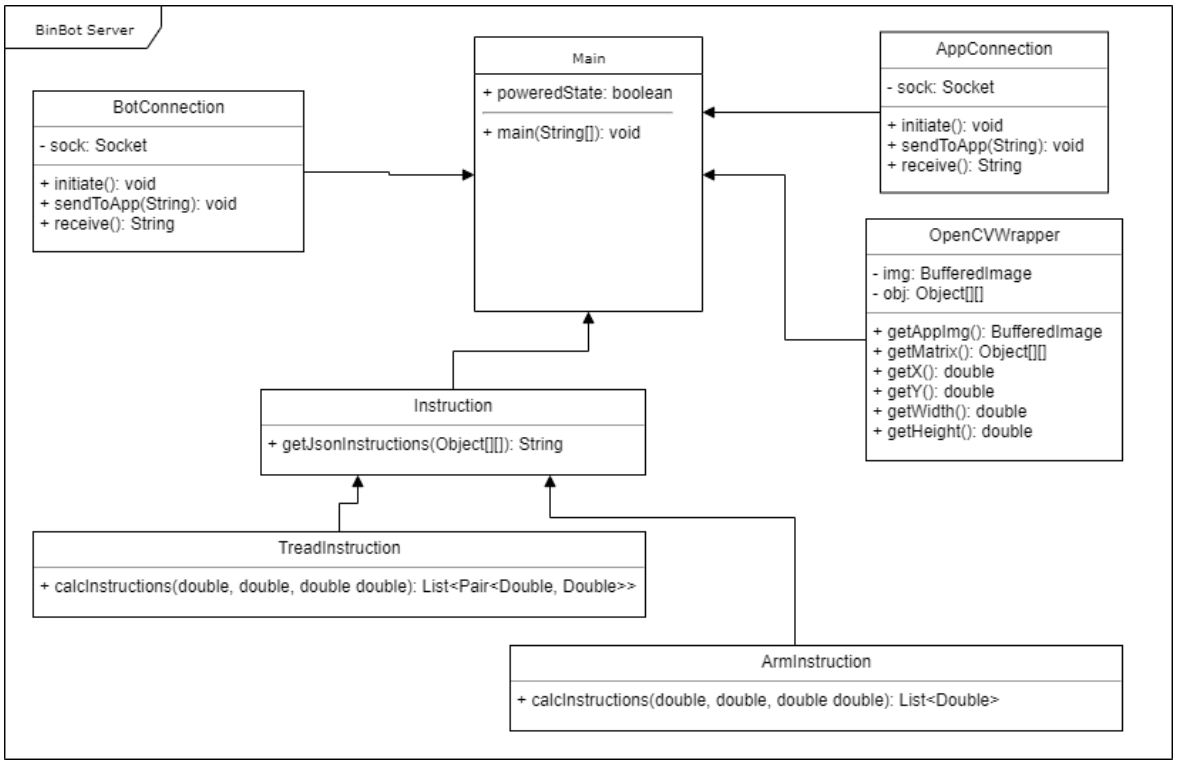

Figure 7: Overall class Diagram/Architecture of BinBot Server

The main class will unify all of these components to the heavy data processing and integration for the BinBot project. The server starts by establishing a connection with both the Arduino board on the robot and the Android operator mobile app via the BotConnection and AppConnection classes. Once the server application has received the start command from the mobile app via the AppConnection class, the server will signal the Arduino to begin the patrolling processing using the BotConnection class. The server will then wait for the robot's image feed, which will be passed to the OpenCVWrapper class.

The OpenCVWrapper class will contain the waste identification functionality using OpenCV based off the training done by the machine learning model. The server will pass the image feed to the class, returning the results of the waste identification on the image. The server will then send the results to the operator mobile app via the AppConnection class. The server will also send the coordinates and size of the bounding boxes formed by OpenCV to the Instruction class.

The Instruction class will attempt to compute the location and size of the waste object based on the OpenCV results. The server will then translate this into instructions for the robot with the TreadInstruction or ArmInstruction classes. The server application will send these instructions back to the Arduino using the BotConnection class.

Communication

The server and BinBot will communicate via JSON strings sent over a socket stream. The JSON object will have the fields *status, img, treads,* and *arms*. Status will hold a string that coordinates with an enumeration representing the different ways the json object should be used. *Img* will hold bitstream of an image taken by the BinBot camera and will only be filled in when images must be sent to the server. *Treads* will be a list of json objects called *Tread* which will hold two integers, *angle*, an integer from –180 to 180 representing how far BinBot should turn before moving forward, and *distance*, a double instructing BinBot how many meters forward it should move after turning. This will be in list format so that multiple *Tread* instructions can be followed by BinBot sequentially. *Arms* will be a list of a*rm* instructions, similar to T*reads,* which contain the *angle*, that each limb should turn. Instead of following each instruction sequentially though, there will always be four instructions with each one corresponding to a specific joint in the arm.

```
Object {
        "status":("PATROL", "NO WASTE", "NAVIGATION", "POST_NAV", "RETRIEVAL"),
        "img":<Bitream>,
        "Treads": [
                {
                        "angle":<Integer>,
                        "distance":<Integer>,
                },
                {
                        "angle:<Integer>,
                        "distance":<Integer>
                }
        ],
        "Arms": [
                {
                        "angle":<Integer>
                },
                {
                        "angle":<Integer>
                }
        ]
}
```

Figure 8: Layout of the JSON object and fields used to communicate between BinBot and the server.

The *status* enumeration will have five possible strings. The first, *PATROL*, will be set and sent to the server by BinBot and will indicate that BinBot is searching for trash. It will be sent along with the *img* field filled so that the server can search for waste to retrieve. If waste is not found, a json object with the status *NO_WASTE* will be sent back to BinBot so BinBot can continue its search and send a new *PATROL* message. Once the server eventually finds waste in the image and calculates how BinBot should navigate to it, it will send back a json object with the status *NAVIGATION* and fill the *treads* object with a list of instructions on how BinBot should move. After BinBot has execute its instructions, it will then send a json message back to the server with the status *POST_NAV*, and a new image. The server will then either send a second *NAVIGATION* message if BinBot's location needs to be corrected or calculate arm movements and send a new message with the status *RETREIVAL*, with the *Arms* object filled with instructions on moving the arm to pick up the trash. After the trash has been retrieved, BinBot will send a new photo without moving and a *PATROL* message, ensuring that the trash has been picked up and resuming patrol.

Mobile Application

The Android mobile application will consist of a single activity, AndroidDev. This activity will include two button views: start, stop, and a fragment to display the image feed. When the app starts, it will use the ServerConnect class to establish a connection to the data processing server. This will allow the app to communicate with the server via the send() and receive() methods. The start button view will have an on touch listener call that, when invoked, will signal the server via the ServerConnect's send() method to start BinBot's waste collection process. The stop button view will invoke a similar functionality, only commanding the server to change to the *standby* state instead. The OpenCV result image will be passed into the app via the ServerConnect receive() method. This image will then be displayed on the screen through the ImageFeedFragment fragment.

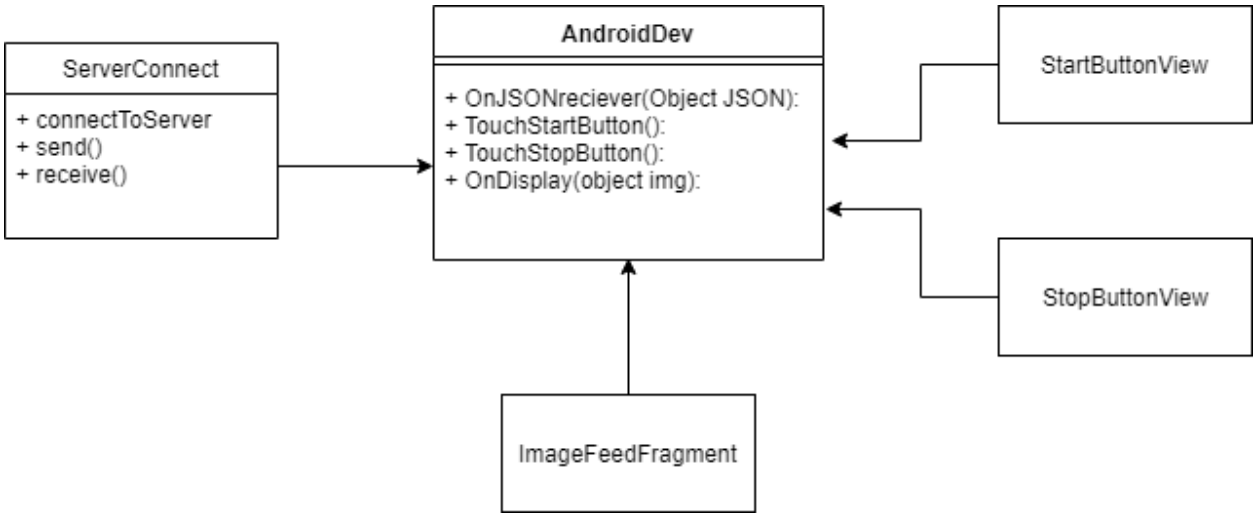

Fig 9: UML diagram of the Android operator mobile application

Communication

The main purpose of having a mobile app in this project is to create an interface for the test users interacting with BinBot. This means the user can be able to pick up their mobile device and activate BinBot's functionality with a touch of a button. To be able to implement this, the mobile application needs to be able to communicate with the main server. This communication will be done with Representational State Transfer (REST). REST is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data. REST API breaks down a transaction to create a series of small modules. Each module addresses a particular underlying part of the transaction. explicitly takes advantage of HTTP methodologies defined by the RFC 2616 protocol. They use GET to retrieve a resource; POST to create or change the state of or update a resource, which can be an object, file or block; and DELETE to remove it.

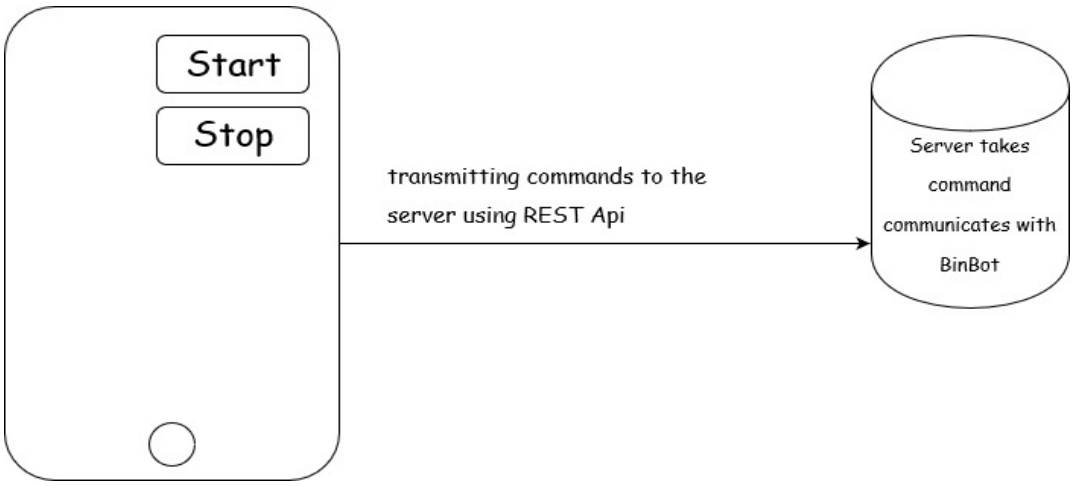

Figure 10: Diagram displaying interaction between mobile application and server

MACHINE LEARNING

The machine learning portion of BinBot appears the simplest for two specific reasons. The first of which is that programmatically, it is largely divorced from the rest of the system. It operates separately from the other real-time components. This will also allow Tensorflow to fully utilize the GPU supported server to train the machine learning model as fast as possible. The output of the machine learning software, the neural network data model, will be in the form of an exported, or "frozen", Tensorflow Protobuf (.pb) file that can be utilized in the server software using the OpenCV library. "Tensorflow models usually have a fairly high number of parameters. Freezing is the process to identify and save just the required ones (graph, weights, etc) into a single file that you can use later" (Vitor par. 3).

The second reason for the appearance of simplicity in the class model (figure 5.1) is that so much of the heavy lifting is done for us by the TensorFlow library. However, despite this appearance of simplicity, it is important to understand what is happening under the covers in order to utilize TensorFlow's machine learning algorithms properly. This requires knowledge of two key concepts, the neural network and tensors.

A tensor is basically another term for a multi-dimensional matrix. In the case of tensors, the number of dimensions, or axes, is referred to as the tensor's "rank". For the sake of our implementation of a training a neural network using images, we will be working with rank four tensors, or four-dimensional matrices. These tensors consist of three-dimensional representations of each image. Each image is represented three-dimensionally by height, width, and color-depth. The fourth dimension of the tensor is for each of these sample images.

> *"There are two conventions for shapes of image tensors. The channels-last convention (used by TensorFlow) and the channels-first convention (used by Theano). The TensorFlow machine-learning framework, from Google, places the color-depth axis at the end: (samples, height, width, color_depth)."* (Chollet 36.)

The second key concept to understand for our implementation is that of the neural network we are training. In the simplest terms, a neural network consists of "layers" that transform the input image tensors from one representation to another until the neural network changes the tensor into a representation it recognizes, or eventually fails to recognize. These layers are basically sets of tensor operations, or multi-dimensional matrix operations. Examples of the transformations used by the neural network are element-wise operations (addition, subtraction, etc), tensor dot (equivalent to the dot product of matrices), broadcasting (increasing the size of the tensor by repeating it over new axis or axes), and tensor reshaping (or matrix

transposition).In the case of training a neural network for image recognition, we are interested in a specific type of neural network called a convolutional neural network, or “convnet”.

The key difference between standard “densely connected” layers of a standard neural network and those of a convnet is that the layers of a convnet are focused on recognizing patterns in small portions of the images. This way, the convnet will be trained to look for small aspects of the objects we wish to recognize such as specific patterns in the color, edges, small patterns, etc. before reaching a decision about the whole. Similarly, this allows the convnet to identify the object or objects within a part of the image, rather than just classifying the whole image as recognized or not, and give specific dimensions of the object’s size within the image so we may display boxes around the recognized objects in the mobile application, and use information about the object’s horizontal and vertical orientation in BinBot’s view to make calculations about the object’s distance and angle from BinBot.

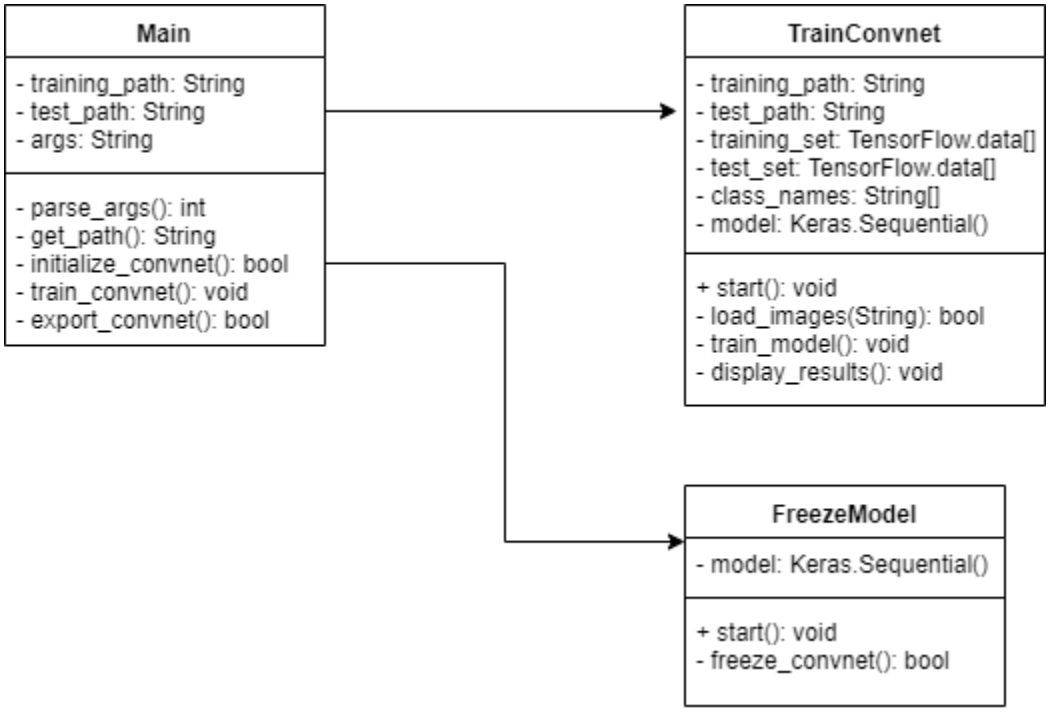

Figure 11: Machine Learning Class Diagram

Algorithmically, the software machine learning implementation is simple. We accept input from the user performing the training in terms of two directories; one containing the images we wish to train the convnet on, and one containing the images we wish to use after training to test the current success level of the trained convnet. These test images are traditionally a mix of

images used for training and unused images from which we expect a positive recognition. These images will consist of both images taken by the BinBot team as well as any provided from TensorFlow. The output of this process is data about the success level of recognition on the training set and the test set, and a comparison of the two. The implementation also contains functionality for the user to initialize the convnet or continue training the existing one, and to output the "frozen" convnet for use in BinBot's server-side OpenCV implementation.

One final note about the implementation is that we will need to be wary of something called "overfitting".

"At the beginning of training, optimization and generalization are correlated: the lower the loss on training data, the lower the loss on test data. While this is happening, your model is said to be underfit: there is still progress to be made; the network hasn't yet modeled all relevant patterns in the training data. But after a certain number of iterations on the training data, generalization stops improving, and validation metrics stall and then begin to degrade: the model is starting to overfit." (Chollet 104.)

There are two easy ways we can handle this with regards to our training images. The first of which is to have plenty of good images that do not contain anything extra in the image that may lead to the model falsely recognizing irrelevant patterns. The second is simply to have a very large set of diverse images, which we plan to collect thousands of using controlled environments and varied lighting.

SEQUENCE DIAGRAM

This sequence diagram describes a use case for BinBot's cycle. The cycle starts with the processing server initializing each component by establishing connections with both the operator mobile app and the Arduino board on the robot. The server waits for the signal to start from the user controlling the app to then signal the robot to begin the patrolling state. The Arduino will capture and send an image back to the server. The server will identify any waste in the image and compute instructions for the robot to collect the target. The server will send these instructions to the robot, repeating the waste collection loop until receiving the signal from the mobile app to stop. The server will relay this signal to the robot, completing the BinBot sequence.

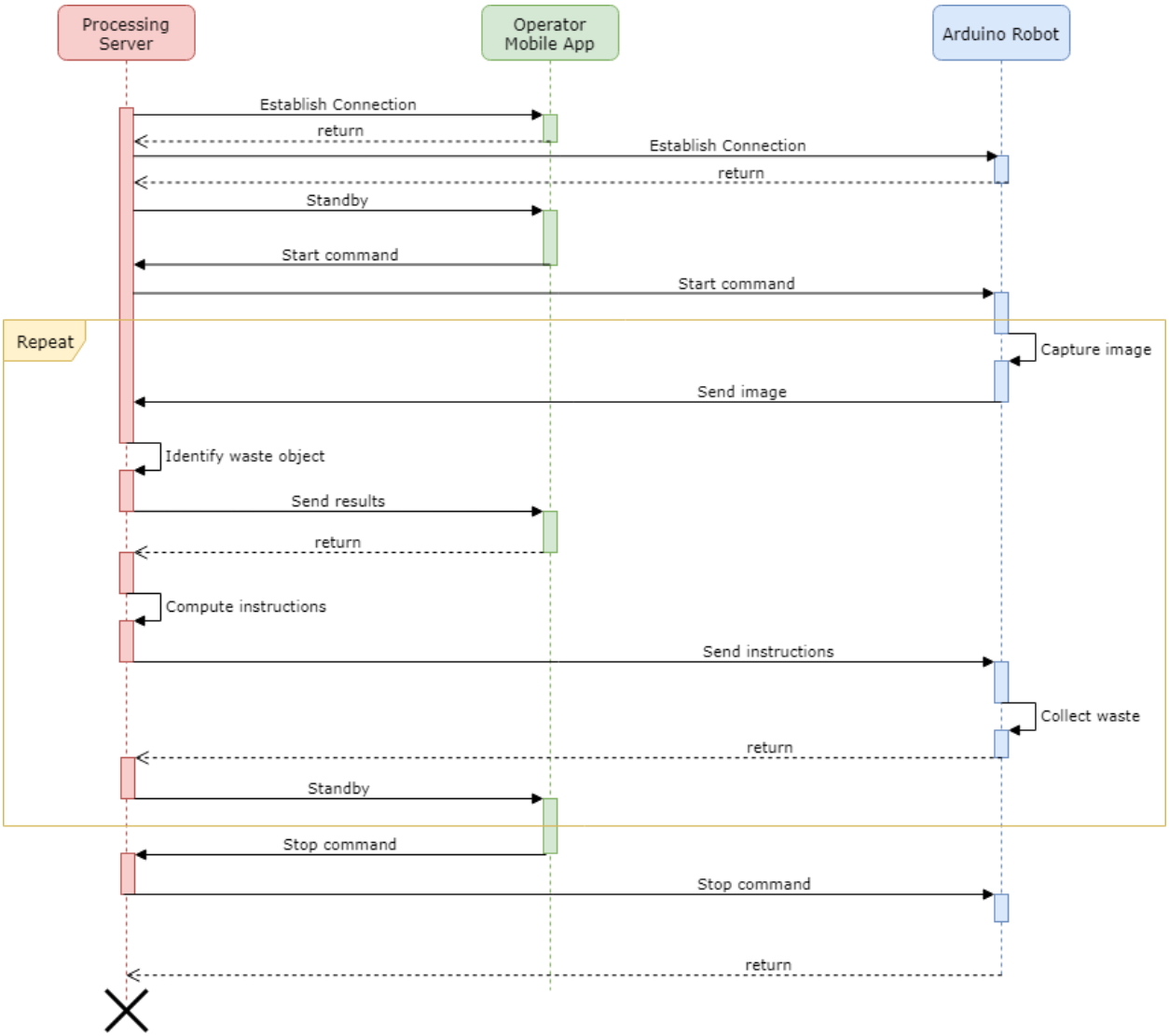

Fig 12: UML sequence diagram describing the flow of BinBot's cycle

REFERENCES

"Arduino Mega 2560 GFS WiFi Video Robot Tank Car Kit with 4 DOf Robot Arm." *Arduino Mega 2560 GFS WiFi Video Robot Tank Car Kit with 4 DOf Robot Arm - XiaoR Geek Official Store*, http://www.xiaorgeek.com/store/arduino-mega-2560-gfs-wifi-video-robot-tank-car-kit-with-4-dof-robot-arm.html.

Chollet, Francois. *Deep Learning with Python*. Manning, 2018. Print.

Vitor, Jean. (2018, October 13). *How to load TensorFlow models with OpenCV*. https://jeanvitor.com/tensorflow-object-detecion-opencv/

Margaret, Rouse. June 2019, A guide to open source technology in application development, RESTful Api. https://searchapparchitecture.techtarget.com/definition/RESTful-API.