BinBot
Design Document II

REVISION HISTORY

Revision #	Author	Revision Date	Comments
1.0	Sean Reddington	October 22, 2019	Copied and updated all of the generated Javadocs
1.1	Michael Savitski	October 24, 2019	Added class descriptions for Machine Learning

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

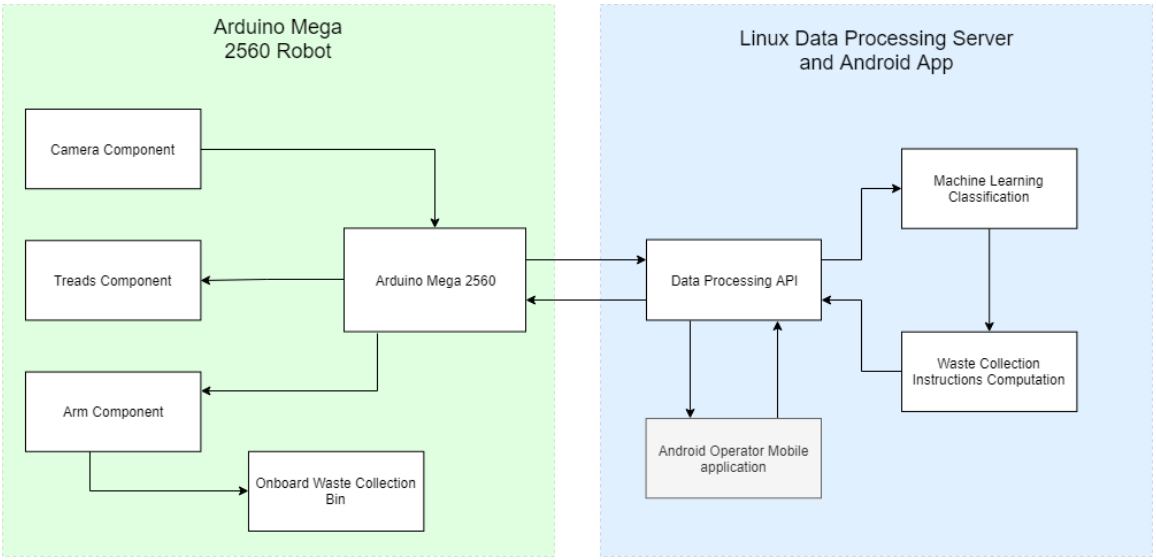

Fig 1. Block diagram illustrates the flow of data from the Camera feed to BinBot disposing of the waste.

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.

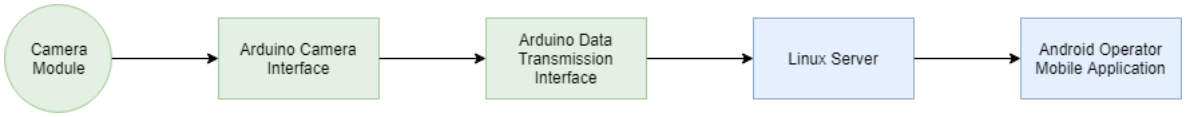

Fig 2. Diagram displaying interaction between BinBot's Camera Module, Linux Server, and Android Application.

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.

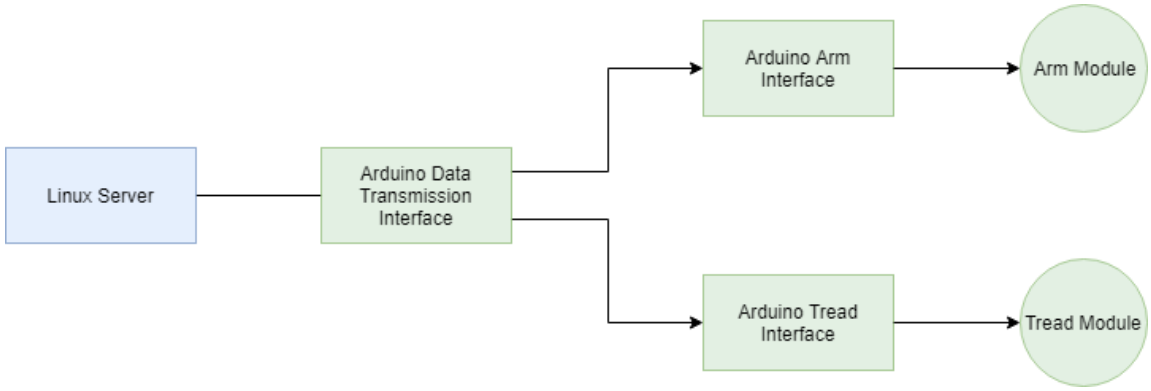

Fig 3. Diagram displaying interaction between the Linux Server and the Arduino's interfaces in order to operate the robot.

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.

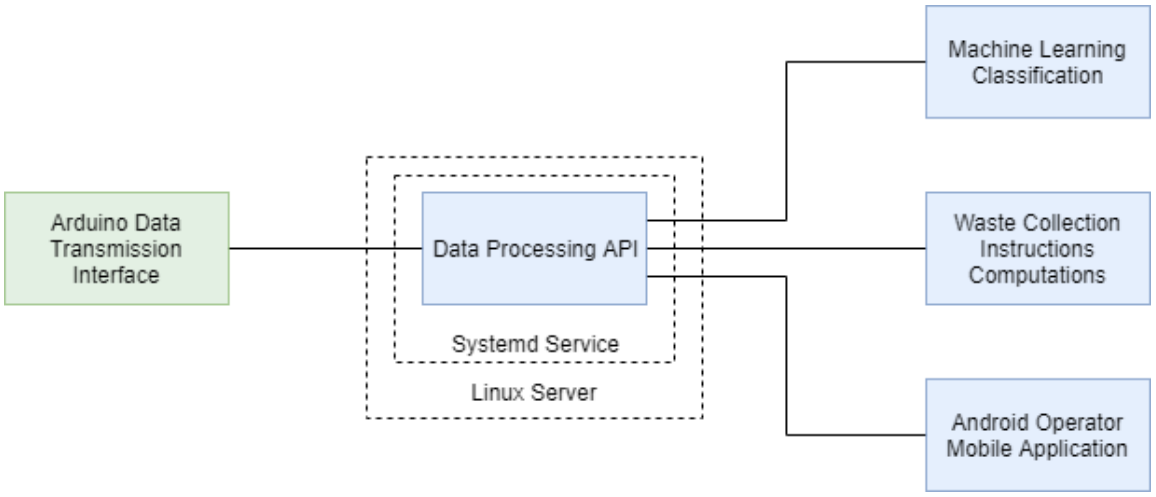

Fig 4. Diagram displaying the interaction between the Arduino's Data Transmission Interface, Linux data processing server, and Android Application.

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.

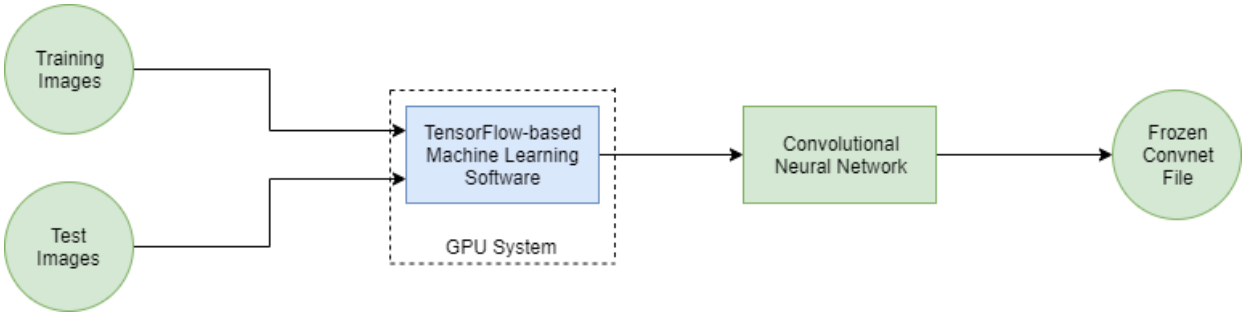

Fig 5. Diagram illustrating the process of training the neural network data model that will enable BinBot to recognize objects.

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.

Arduino Sketch

Class Main

Main class for the Arduino Sketch loaded onto the BinBot robot kit's Arduino board.
Author: *Jose Silva*
Version: *1.0*
Since: *2019-10-13*

void setup()

This method establishes any prerequisite configurations needed for BinBot's robotic loop to function. This method is called immediately once the Arduino is in a powered state.
Author: *Sean Reddington*
Version: *1.0*
Since: *2019-10-13*

void loop()

This method is continuously called throughout BinBot's Arduino's powered state. This method will take and send an image feed to the data processing server and wait to receive instructions on collecting any waste objects back. If instructions are received, BinBot will invoke the necessary interfaces in order to collect the waste object.
Author: *Sean Reddington*
Version: *1.0*
Since: *2019-10-13*

Interface Transmission

Data transmission interface to allow communication between the BinBot Arduino board and the data processing server.
Author: *Sean Reddington*
Version: *1.0*
Since: *2019-10-13*

boolean network_connect(String network_host)

This method establishes a connection to the network which hosts the data processing server, e.g., Temple University's wireless network.
Param: *network_host*
Return: *Boolean stating the results of the attempted connection*
Author: *Sean Reddington*
Version: *1.0*
Since: *2019-10-13*

boolean server_connect(String server_host)

This method establishes a connection channel between the client and a host server given the server's host name. This will allow the client to exchange data with the server.
Param: *server_host*
Return: *Boolean stating the results of the attempted connection*
Author: *Sean Reddington*
Version: *1.0*
Since: *2019-10-13*

boolean send(String data, String server)

This method attempts to send the passed data to the connected server.
Param: *data*
Param: *server*
Return: *Boolean stating the results of the attempted transmission*
Author: *Sean Reddington*
Version: *1.0*
Since: *2019-10-13*

String receive(String server)

This method attempts to receive data from the connected server.
Param: *server*
Return: *String containing data received from server*
Author: *Sean Reddington*
Version: *1.0*
Since: *2019-10-13*

String pack_json(JsonObject json)

This method will take the passed JsonObject and attempt to convert it to a string.
Param: json
Return: String
Author: Sean Reddington
Version: 1.0
Since: 2019-10-13

JsonObject unpack_json(String data)

This method will take the passed data string and attempt to convert it to a JsonObject (JavaScript Object Notation) object.
Param: data
Param: JsonObject
Return: Boolean stating the results of the attempted transmission
Author: Sean Reddington
Version: 1.0
Since: 2019-10-13

Interface Treads

Treads interface to control robot kit's physical tread components.
Author: Jose Silva
Version: 1.0
Since: 2019-10-13

void patrol()

This method is used to allow BinBot to move a set amount of distance. Calling this method will instruct BinBot to travel a set distance in its search for waste objects.
Author: Jose Silva
Version: 1.0
Since: 2019-10-13

void forward()

This method is used to invoke the treads on BinBot. Calling this method will make BinBot move in a forward direction.
Author: Jose Silva
Version: 1.0
Since: 2019-10-13

void leftTurn()

This method is used to invoke the treads on BinBot. Calling this method will make BinBot's treads make a left turn.
Author: Jose Silva
Version: 1.0
Since: 2019-10-13

void rightTurn()

This method is used to invoke the treads on BinBot. Calling this method will make BinBot's treads make a right turn.
Author: Jose Silva
Version: 1.0
Since: 2019-10-13

void stop()

This method is used to halt the treads on BinBot. Calling this method will bring BinBot to a standstill by ceasing the treads.
Author: Jose Silva
Version: 1.0
Since: 2019-10-13

Interface Arm

Arm interface to control robot kit's physical arm component.
Author: *Jose Silva*
Version: *1.0*
Since: *2019-10-13*

void shoulder()

This method is used to invoke the shoulder mechanism built into the Mega 2560 GFS robot tank's arm.
Author: *Jose Silva*
Version: *1.0*
Since: *2019-10-13*

void elbow()

This method is used to invoke the elbow mechanism of the Mega 2560 GFS robot tank's arm.
Author: *Jose Silva*
Version: *1.0*
Since: *2019-10-13*

void wrist()

This method is used to invoke the wrist mechanism of the Mega 2560 GFS robot tank's arm.
Author: *Jose Silva*
Version: *1.0*
Since: *2019-10-13*

void claw()

This method is used to invoke the claw mechanism of the Mega 2560 GFS robot tank's arm.
Author: *Jose Silva*
Version: *1.0*
Since: *2019-10-13*

Interface Camera

Camera interface to control robot kit's physical camera components.
Author: Jose Silva
Version: 1.0
Since: 2019-10-13

void capture_photo()

This method invokes the camera feature on BinBot; taking a photo of what is in front of its path.
Author: Jose Silva
Version: 1.0
Since: 2019-10-13

double get_proximity()

This method is used to calculate the distance to an identified object.
Return: Double value representing the distance between the sensor and an object
Author: Jose Silva
Version: 1.0
Since: 2019-10-13

boolean in_range()

This method is used to check if an identified waste object is in range of BinBot's arm, if the waste is in range, the method will return true otherwise it will return false.
Return: Boolean representing if the object is in range of arm
Author: Jose Silva
Version: 1.0
Since: 2019-10-13

Machine Learning

Module main

Main module to execute waste object identification machine learning.
Author: *Michael Savitski*
Version: *1.0*
Since: *2019-10-12*

main()

Main method ran at start up.
Author: *Michael Savitski*
Version: *1.0*
Since: *2019-10-12*

parse_args(args)

This method parses the command line arguments from starting the application and returns them to the main function.
Param: *args: The arguments passed in from the command prompt*
Return: *args*
Author: *Michael Savitski*
Version: *1.0*
Since: *2019-10-12*

initialize_convnet(model_path)

This method will load the existing convnet or initialize it if none exists.
Param: *model_path: The expected file path to the existing model*
Return: *args*
Author: *Michael Savitski*
Version: *1.0*
Since: *2019-10-12*

train_convnet(training_path, test_path, model)

This method will begin the training of the convnet by calling the TrainConvnet class
Param: *training_path: file path to the set of training images*
Param: *test_path: file path to the set of test images*
Author: *Michael Savitski*
Version: *1.0*
Since: *2019-10-12*

export_convnet(model)
This method will export the model to a file by calling the FreezeModel class
Param: model: the convnet data model
Author: Michael Savitski
Version: 1.0
Since: 2019-10-12

Class FreezeModel

Class that is responsible for "freezing" the neural network data model into a file format that can be used by OpenCV.
Author: Michael Savitski
Version: 1.0
Since: 2019-10-12

start(self)
This method will be called externally by main to begin exporting the model to a file.
Param: self
Author: Michael Savitski
Version: 1.0
Since: 2019-10-12

Class TrainConvnet(training_path, test_path, model)

Class responsible for executing the training loop of the convnet neural network using the images designated by the user.
Param: training_path: Path to the directory containing the training images.
Param: test_path: Path to the directory containing the test images.
Param: model: Convnet data model to be trained on, passed in from the main module.
Author: Michael Savitski
Version: 1.0
Since: 2019-10-12

start(self)
This method will be called externally by main to begin training the model.
Param: self
Author: Michael Savitski
Version: 1.0
Since: 2019-10-12

Operator Mobile App

Class AndroidDev

Main class for operator Android application
Author: Kwamina M Thompson
Version: 1.0
Since: 2019-10-13

OnJSONreceiver(Object JSON)

This method will receive all JSON objects and translate it to formats that can be read in Java/Android system.
Author: Kwamina M Thompson
Version: 1.0
Since: 2019-10-13

TouchOnButton()

This method gets called when a user touches the on button and sends packets to the server to turn on BinBot
Author: Kwamina M Thompson
Version: 1.0
Since: 2019-10-13

TouchOffButton()

This method gets called when the user touches the off button on the screen and notifies the server to shutdown BinBot's operations.
Author: Kwamina M Thompson
Version: 1.0
Since: 2019-10-13

OnDisplay(Object img)

This method takes in an image object that has been processed by the OnJSONReceiver method, then displays the image on the application's screen.
Param: img: image object
Author: Kwamina M Thompson
Version: 1.0
Since: 2019-10-13

Processing

Interface AppConnection

The AppConnection class represents the network connection between the BinBot Mobile Application and the BinBot Server. Its methods allow the server to initiate and wait for a connection to be established, and then once it is established, data can be sent back and forth to the mobile application over a socket.
Author: Sean DiGirolamo
Version: 1.0
Since: 2019-10-13

void sendToApp(InputStream in)

This method takes as input an InputStream which will be sent over the socket to whatever client is connected to it, in this case the mobile application.
Param: in: InputStream from socket
Author: Sean DiGirolamo
Version: 1.0
Since: 2019-10-11

OutputStream receive()

This method instructs the server to wait to receive a bitstream from the client. This bitstream will be returned to the caller as an OutputStream.
Return: OutputStream: bitstream from client
Author: Sean DiGirolamo
Version: 1.0
Since: 2019-10-11

void initiate()

This method should be the very first method called by this object. It initiates the socket connection and doesn't complete until a client has connected. This must happen before any other methods or data transfers can complete.
Author: Sean DiGirolamo
Version: 1.0
Since: 2019-10-11

Interface BotConnection

The BotConnection class represents the network connection between the BinBot robot and the BinBot server. Its methods allow the server to initiate and wait for a connection to be established and then once it is established, data can be sent back and forth to the robot over a socket.
Author: Sean DiGirolamo
Version: 1.0
Since: 2019-10-13

void sendToBot(InputStream in)

This method takes as input an InputStream which will be sent over the socket to whatever client is connected to it, in this case the BinBot robot.
Param: in: InputStream from socket
Author: Sean DiGirolamo
Version: 1.0
Since: 2019-10-11

OutputStream receive()

This method instructs the server to wait to receive a bitstream from the client. This bitstream will be returned to the caller as an OutputStream.
Return: OutputStream: bitstream from client
Author: Sean DiGirolamo
Version: 1.0
Since: 2019-10-11

void initiate()

This method should be the very first method called by this object. It initiates the socket connection and doesn't complete until a client has connected. This must happen before any other methods or data transfers can complete.
Author: Sean DiGirolamo
Version: 1.0
Since: 2019-10-11

Interface ArmInstruction

The ArmInstruction class consists only of one static method "calcInstructions", which is responsible for calculating the movements BinBot should takes to retrieve a piece of waste using the joints in its arm.
Author: *Sean DiGirolamo*
Version: *1.0*
Since: *2019-10-13*

List<Double> calcInstructions(double x, double y, double w, double h)

This method takes as input, an x coordinate, a y coordinate, a width, and a height, all indicating the location of an identified object's bounding box in a matrix. Using these values, it will estimate a distance between the camera and the object, and the angle that the camera is viewing the object at. The function will then generate a list of movements that should be applied to the arm's joints, such that the arm will be in a position where it would grab the object were it to squeeze. These instructions will be in the form of a list of doubles, each double indicating the angle their corresponding joint should rotate. This list is returned to the caller.
Param: *x: x coordinate of bounding box*
Param: *y: y coordinate of bounding box*
Param: *w: width of bounding box*
Param: *h: height of bounding box*
Return: *List of doubles, indicating the angle their corresponding joint should rotate*
Author: *Sean DiGirolamo*
Version: *1.0*
Since: *2019-10-09*

Interface TreadInstruction

The TreadInstruction class consists only of one static method "calcInstructions", which is responsible for calculating the movements BinBot should take to navigate to a piece of waste in regard to its treads.
Author: Sean DiGirolamo
Version: 1.0
Since: 2019-10-13

List<Pair<Double, Double>> calcInstructions(double x, double y, double w, double h)

This method takes as input, an x coordinate, a y coordinate, a width, and a height, all indicating the location of an identified object's bounding box in a matrix. Using these values, it will estimate a distance between the camera and the object, and the angle that the camera is viewing the object at. The function will then generate a list of movements that should be taken to navigate to the object, as angle-direction pairs, meant to indicate first a turn, then a direction traveled straight. This list will be returned to the caller.
Param: x: x coordinate of bounding box
Param: y: y coordinate of bounding box
Param: w: width of bounding box
Param: h: height of bounding box
Return: list of movements to navigate to the object, as angle-direction pairs
Author: Sean DiGirolamo
Version: 1.0
Since: 2019-10-09

Interface Instruction

The Instruction class consists of a single static method which is responsible for providing a JSON string which can be used to properly maneuver BinBot to retrieve waste when given a matrix from OpenCV. More details can be found in the method documentation itself.
Author: *Sean DiGirolamo*
Version: *1.0*
Since: *2019-10-13*

String getJsonInstructions(Object[][] matrix)

This method takes as input, a matrix of objects generated from OpenCV computer vision and returns a json string of the format:

```
{
   "status":,
   "treads":[
        {
        "angle":,
        "momentum",
        }
   ],
   "arms":[
        {
         "angle":
        }
   ]
}
```

This JSON string is meant to be sent to the BinBot robot to instruct the robot what movements it should take. Within this function, the server will decide how BinBot should proceed based on the matrix it is provided. The matrix will first be used to calculate the distance the object is predicted to be from the BinBot and its angle. If the waste is too far, the function will decide that the robot must move to the waste and will generate a list of "treads" instructions to tell the robot how it should move. If it is decided that the robot is close enough, instead of generating a list of tread instructions, a list of arm instructions will be generated, instructing each limb how they should move to retrieve the waste. Either way, these instructions will be packed into a JSON string and returned to the caller.
Param: *matrix of objects generated from OpenCV*
Return: *json string of instructions*
Author: *Sean DiGirolamo*
Version: *1.0*
Since: *2019-10-09*

Interface OpenCVWrapper

The OpenCVWrapper class serves as an easy to use front end for initiating OpenCV on an image and for retrieving data about the image, if waste has been located, where it has been located, its height, its width, and the matrix representation of that image itself.
Author: *Sean DiGirolamo*
Version: *1.0*
Since: *2019-10-13*

BufferedImage getAppImg()

This method returns a BufferedImage, which, if waste has been detected in it, will have a box surrounding that waste.
Return: *BufferedImage*
Author: *Sean DiGirolamo*
Version: *1.0*
Since: *2019-10-11*

Object[][][] getMatrix()

This method returns an Object matrix which was generated by OpenCV. This matrix represents the image and identification of a waste object and its location.
Return: *Object matrix generated by OpenCV*
Author: *Sean DiGirolamo*
Version: *1.0*
Since: *2019-10-11*

double getX()

This method returns the x value as a double of the origin of the section of the image identified to contain waste.
Return: *x value of bounding box*
Author: *Sean DiGirolamo*
Version: *1.0*
Since: *2019-10-11*

double getY()

This method returns the y value as a double of the origin of the section of the image identified to contain waste.
Return: *y value of bounding box*
Author: *Sean DiGirolamo*
Version: *1.0*
Since: *2019-10-11*

double getHeight()

This method returns the height value as a double of the section of the image identified to contain waste.
Return: height value of bounding box
Author: Sean DiGirolamo
Version: 1.0
Since: 2019-10-11

double getWidth()

This method returns the width value as a double of the section of the image identified to contain waste.
Return: width value of bounding box
Author: Sean DiGirolamo
Version: 1.0
Since: 2019-10-11