General Interactive Automation Network

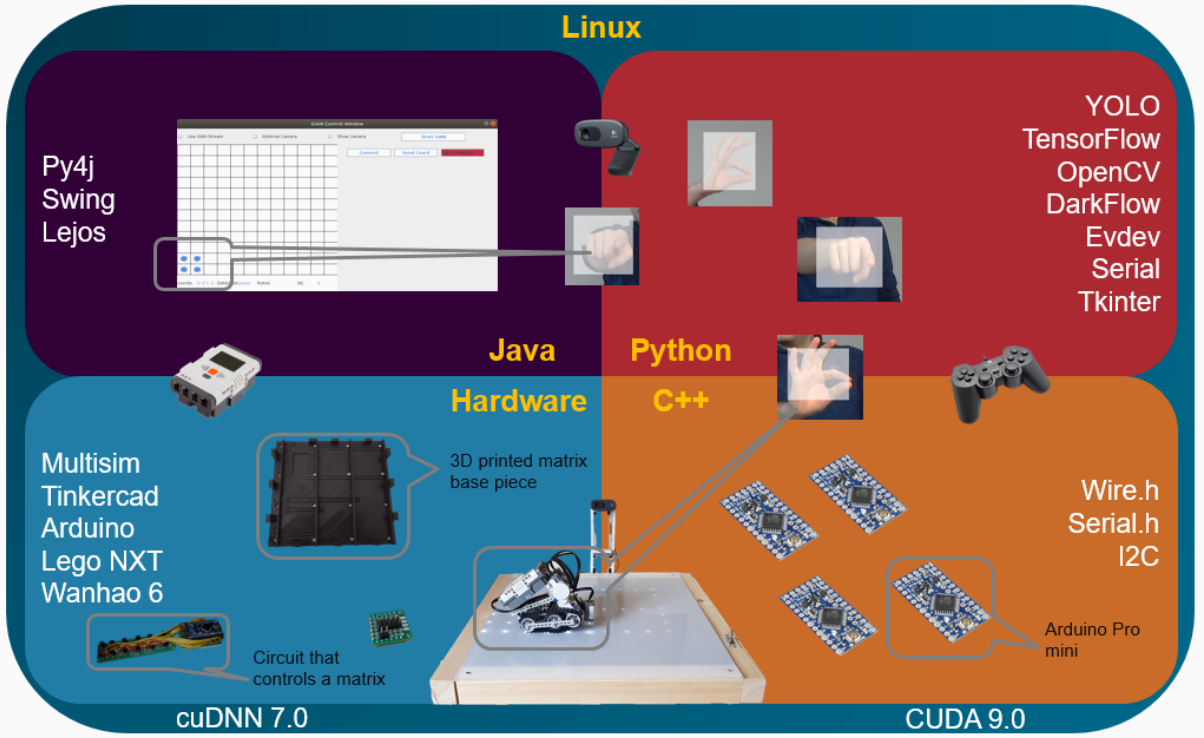
Name: Vahe Grigoryan

ID: 1600567

Course: Computer Systems Engineering

Supervisor: Dr. Manoj Thakur

Second assessor: Dr. Martin Reed







Acknowledgements

It is a pleasure to remind the fine people in the University of Essex for their help.

I would like to express my appreciation to Dr. Manoj Thakur for his guidance and feedback during the course of the project.

A special thanks to Workshop Technician/Toolmaker, Paul Vincent for making a beautiful wooden frame for the project components.

Thank you very much for your guidance and efforts.

Summary

With growing number of Internet of Things “IoT” devices almost everything in our homes and offices can be automated or controlled remotely. The General Interactive Automation Network “GIAN” is a system that is trying to achieve full home automation, while providing a simple gesture based user interface. GIAN uses simple gestures to select and operate devices with various physical and technical capabilities. A small scale model of the system is developed to test its practicability and limitations. In this report we will discuss the existing legal and ethical issues regarding to similar systems, cover the technologies used for building it and reflect on project planning technologies and methodologies used during the course of the project.

Table of Contents

List of Symbols

GIAN – General Interactive Automation Network

(G)UI – (Graphical) User Interface

IoT – Internet of Things

LED – Light Emitting Diode

SDK – Software Development Kit

CPU – Central programming unit

GPU – Graphics processing unit

RAM – Random access memory

OS – Operating System

JVM – Java Virtual Machine

IDE – Integrated development environment

TCP – Transmission Control Protocol

IP – Internet Protocol

USB – Universal serial bus

CNN – Convolutional Neural Network

1 The Problem

1.1 Description

There are many smart devices produced by different companies, which means in order to use them users have to learn different commands and get used to different user interfaces (UI). According to this "NetworkWorld" [1] article elderly people find it very difficult to use such systems, because of the accessibility and complex UI designs. This issue is an important one because a lot of IoTs are integrated into costumer medical equipment and ordinary home furniture. Being able to easily control such devices can be very beneficial and life saving for many people. Unfortunately there are no such systems that have deep integrations with IoTs with simple user interfaces.

IFTTT [2] is a system that tries to solve this problem, but it is dependent on Google online services and users ability to work with their online system. User has to create an account to use online services with IoT devices implementing "if this than that" logic, for someone with limited knowledge in commuters, this could be a difficult task.

Most solutions in this field use more or less the same approach.

1.2 Legal and ethical issues

Having devices developed with different companies raises an issue of getting the rights to integrate a third party software with the one that the company has produced. Likely most companies provide SDKs (Software Development Kit) for their product, which can be used to control the product to a certain extent with custom software. Usually some sort of license will be associated with the code developed with a SDK depending on the scale of the integration. It is important to be careful when working with different SDKs to avoid any legal problems.

Having a smart home can be very beneficial in terms of operating home appliances, being more efficient with heating and cooling and more, however there are ethical issues associated with devices tracking residences behaviour. Some of them are storing user data on an online database so that user can have access to their data everywhere they go. In situations like this all the user has is the trust in the company to not sell or use the collected data without their knowledge. If the collected data is misused it could lead to violations of user's privacy and even to identity theft. It is important to know who stakeholders are and how their privacy can be protected when developing a product that interacts with users and collects data about them.

As stated in this ieee [3] article there are range of technological challenges associated with IoT devices and security is one of them. Hacking an IoT device is interesting to malicious users because of their vulnerability and the variety of information they contain about a user. The main weakness of these devices is the internet connectivity they have, which allows hackers to get access to them and once the device is compromised a lot of private information can be obtained.

1.3 Sustainability

Solutions similar to IFTTT depend on online services that require the user to have internet connection, which could be a deal breaker for the user who does not want to upload their information to the internet. To sustain such systems they have to make sure that all components can be synced online and any updates to different parts of the system would not affect the integrity of the whole system, which is of course hard due to external factors.

Ideally an automated home should have a command centre or a small server of a sort that can manage all smart devices while running locally and without needing to be connected to external networks. Sustaining such system is easier, because everything is local and there will be no unexpected updates that could potentially break the system.

2 The Solution

2.1 Description

The aim is to create a simple gesture based user interface, which can use local networks (eg. Wi-Fi, Bluetooth, wired and more) to communicate with devices that are within the systems reaching range, while not relaying on external online technology.

In the scope of this project a general interactive automation network was developed to allow users to control a robot with two hand gestures on a custom made LED matrix that provides useful information about users commands. In this context the robot resembles a sofa or other IoT enabled heavy object, which is operated by the user and the LED matrix is the floor of a room. The other features such as controlling the matrix, connecting to a robot and a smartphone are developed to demonstrate the scalability of the system and its ability to connect to multiple devices and manage communication between them at the same time.

The control method of GIAN can be especially practical when connected to IoTs that are integrated in ordinary objects such as armchairs, tables or other heavy furniture allowing disabled users operate those more easily.

2.2 Requirements

The decisions made about what technologies must be used are based on research that is described in the “Initial Report” that can be found here [4].

The system consists of two major parts the software and the hardware. Those parts are made up of many different technologies that come together to provide the underlying support to establish and manage the communication between them. Some of these technologies are inherited from different open source or free software and libraries developed by different universities and organisations.

The following resources are required to setup and run the system.

- Hardware
  - Computer
    - RAM >= 8 GB (min)
    - CPU >= 3.0 GHz (i5, i7 or similar)
    - GPU = NVIDIA Card with compute capabilities grader than 3.0.
    - OS = UBUNTU 18.0
  - Lego NXT
  - Arduio UNO
  - 3 Arduino Pro Minis
  - 3D printer (optional)
- Software
  - Nvidia graphics card driver (GPU specific)
  - CUDA 9.0 [5]
  - Cudnn 7.0 [6]
  - Python 3.x.x
    - TensorFlow-gpu 1.12
    - DarkFlow [7]
    - YOLO9000 [8]
    - OpenCV 3
    - Evdev
    - Tkinter
    - Serial
    - Py4j
  - Java
    - LejOS
    - Py4j
  - Arduino IDE
  - C/C++
    - Serial
    - Wire.h

2.3 Intellectual property

YOLO is a Neural Network model for object detection, which is built using DarkFlow, by University of Washington. It is trained for 9000 object, but unfortunately it is not trained for hand gestures.

A neural network model used in this project is trained for two hand gestures, but underlying technology is that of darkFlow and Yolo.

2.4 Software

2.4.1 Preparing the environment

An appropriate working environment has to be created for the system to work properly.

The very first thing we have to do is to install the GPU driver and CUDA library on a computer. The CUDA is developed by NVidia to optimise their GPUs for machine learning tasks. The installation of CUDA is fairly easy process and is well documented in the official website [5], but a problem might occur, if the GPU driver is installed from the CUDA installer, causing the operating system to not login or computer might not start the OS all together. To avoid this problem the driver can be installed manually. The right driver for a GPU can be found on NVidia web site.

Once CUDA is installed, the Cudnn has to be downloaded and the obtained files have to be placed in the corresponding folders where CUDA is installed. This is important, because this library optimises the GPU for neural networks.

Now we have to install all the python, Java and C/C++ dependencies.

The easiest way to install libraries listed in **Requirements** section is as follows:

- PIP can be used for installing python libraries except DarkFlow and YOLO (one can install those manually, but it will be easier to clone them from the project GitLab [9]).
- Maven can be used for installing java libraries
- Arduino IDE can be used for installing c/c++ libraries.

2.4.2 Training a YOLO model

The gesture detection happens on computer's GPU which is achieve by using "DarkFlow" that runs on "TensorFlow GPU" library. "DarkFlow" is a platform that is used for designing neural networks and "YOLO" is a convolutional neural network that is using it and can do real time object detection. The whole design of the network is stored in a ".cfg" file. The file contains blocks of text where each block is a description of a network layer. The "YOLO" model we using has 17 layers 9 of which are "convolutional" layers, 6 are used for "maxpoolling", one is a "net" layer that describes the network's size, momentum, channels etc. and the last layer is responsible for producing the results. This layer had to be modified to work for this project. Since we are going to train this network for two objects (*"OK"-gesture and "Pointing to camera with index finger"-gesture*) we have to specify that the number of output classes is going to be 2 and we have to select an activation function, which in this case is going to be "SoftMax" because we want to have unique outputs.

The network is trained on 900 images where both gestures have equal number of image 450 each. The reason for such small dataset is the data collection and labelling process, which is very time consuming. For every image an "xml" file had to be created that contains the details of where the hand is on an image and what gesture it is showing. No external datasets with this type of labelling were found that could be used in this project. Likely there is a free software "LabelImg" [10] that makes the job of labelling images a bit easier.

The Linux script "trainYolo.sh" was written for training the model. It takes two arguments the first one must be a directory of "xml" files, which are the annotations of the collected images and the second argument is the directory of the images. The images and according "xml" files must have the same name (e.g. "img1.png", "img1.xml"). By default the script is going to train the model on CPU, which will be painfully long process, to enable GPU the following (--GPU="1.0") must be added to the end of the script. The specified number can be anything between 0.0 and 1.0

it just specifies how much of the GPU is to be used.

Even though the dataset is a small one, but the model did perform better than expected, which makes the system usable at this stage with about 90% accuracy.

2.4.3 Python code

The python code is responsible for detecting hand gestures, estimating the hand position, receiving commands from a Joystick, transmitting the coordinates of detected hand and gesture to Arduino and to JAVA virtual machine (JVM).

Since many different tasks have to be programmed to work together we have to make sure that the individual technologies are working great on their own and we have to come up with a structure enabling those technologies to seamlessly work together.

Every task that does something unique is developed as a class and can be easily testes separately if necessary. Every class is written in a file of their own along with some "commented" code to test their various functionalities, in addition to this unit tests are written to test classes when small changes are made to avoid running them separately every time.

The python program is composed of 6 classes one of, which is an organizer class. This class combines the necessary bits from all classes after synchronising them to achieve the desired result. There is a class that opens a camera and captures frames from it on a separate thread, another class takes those frames and using the computers GPU tries to find the hand gesture and its position, the Java and Arduino communicator classes take the estimated data to send them to JVM and Arduino boards accordingly.

If user decides to not use the camera different class can be used for receiving commands from a joystick, in which case the detector and the camera classes will not be used and the hand positions will be obtained from the joystick class.

Finally the organizer class makes sure that the classes that run on separate threads are synchronised when collecting data, furthermore it implements a logic that allow users to direct information from one class to the other via hand gestures.

There is another class, which is responsible for starting the other classes in a right order when it is executed. This class is needed because the python code can be started from JVM or it can run by itself, so for each of these cases a different configuration is required. When the class is started from Java code we need to use java communicator class to open a bridge between python and JVM, but when it is ran from console no communication with JVM is needed so the class will not be initialised. This feature is added to make the debugging for python easier, since when it is ran from java the user does not get error messages about python code.

2.4.3.1 More details about the classes

Class	**File**
OpenCamera	CameraIO.py
Detector	YoloDetector.py
SerialToArduino	arduino_communicator.py
CommOrg	communication_organiser.py
JIO	java_communicator.py
JoystickIO	joystick_communicator.py

Figure 1 - Python classes and files

OpenCamera class uses "tkinter" library to create dialog box with two camera options for user to select, internal (computer's built in camera) or an external camera (connected via USB). "OpenCV" library is used for turning on a camera based on the user selection and on a new thread frames are captured from the camera. Finally there is a method called "getFrame" which returns the last captured frame.

Detector class is responsible for taking frames from "OpenCamera" class and doing hand gesture detection from them. In addition to that it also finds the hand's location on the frame and converts it to (12x12) coordinate space. The constructor of "Detector" class creates necessary thread locks and global variables to get everything synchronised when needed. It also has a setup method that takes an "OpenCamera" object to be used for obtaining camera footage and initialises two threads, one for detecting gestures and one for displaying them on a new window. The class itself extends to "Thread" class, which means it has a "run" method that runs on a separate thread and it is used for finding the position of the user's hand and converting it to (12x12) coordinate system (*For LED matrix*).

The YOLO model produces a dictionary when an object is detected. The dictionary contains the label ("gesture type"), the confidence of the network, bottom right and top left coordinates of the hand. We are interested in label and the hand coordinates.

There are multiple ways to convert the hands position to (12x12) coordinate space. In version one of the system a mapping was used, so the width of the image was mapped to values from 0 to 11, which is the x coordinate of the hand and the y coordinate was calculated by drawing a bounding box around the hand and calculating its area. If the area is large it means the hand is close to the camera, the opposite is true if area is small and these values are of course in range 0 to 11. This approach worked quite well, but it had a problem, sometimes the area of the bounding box would get large when the hand is not close to the camera, this would usually happen when the user tried to show the "OK" sign mainly because this sign occupies more space that the "index finger" sign, which makes the system think that the hand is closer to the camera and that affects the experience causing confusions to the users, so a better solution was needed.

The new approach involves dividing the image into 144 pieces where each piece corresponds to a specific coordinate, for example the first piece is (0, 0) and the 144^{th} is (11, 11). The hand cannot fit in one image piece so we have to have an additional logic to determine which piece to choose. The method "chooseTheRightAngle" does this by looking at the hands position and based on to what side the hand is shifted toward a selection is made and the coordinate is returned. For example if the hand is closer to the bottom of the image and is shifted towards right than the coordinates of bottom right piece, which is covered by the hand will be returned. This method works better and is consistent, thus it is more reliable and pleasant to use.

Using the detector requires a lot of resources and it becomes hard to debug other features of the system, this is the reason why a joystick is added to the system.

JoystickIO class allows user to produce coordinates similar to one returned from "Detector" class. This class uses "Evdev" library to listen to joystick events on a new thread, it also has a "getCoord" method that returns last values captured from joysticks x and y axis. The class also has a "transform" method that uses the mapping approach from "Detector" class to fit the joystick coordinates in the range 0 to 11, this method will work fine here because the values produced by the joystick are consistent.

SerialToArduino class opens computer's serial port (USB/Serial) and looks for an arduino board using "Serial" library. If board is connected it will start a serial communication with it. The class has multiple methods, which allow other classes to transmit coordinates to Arduino boards. The "showCoords" method is the most important one, this method takes hand position and robot's (*can be any other IOT device*) position to calculate the coordinates of LEDs that have to be turned on, which are than transmitted to Arduino. The method uses "make_robot_occ_space" method which is specific to the robot used for

this project, but it can be modified to work for other robots with different dimensions. This method is important because we have to show the area that the robot occupies to user so they know if the robot is going to fit in the location they are pointing at.

JIO class enables communication between python and java, it uses "py4j" library to do so.

The way this works is as follows:

- Set of memory locations are reserved
- A Java "Stack" object is stored on the memory
- A Java "Map" object is pushed to the "stack".
- The Map is then updated from both programming languages

The "py4j" library allows python to work with Java objects.

JIO class just handles the popping and pushing of the map. It has a "pop_from_stack" method that returns a map object, if no map is present in the stack it returns the last map it popped.

The "push_to_stack" method takes a map object and pushes it to the stack, if there is a map in the stack it replaces it, the stack always has only one map in it.

The "get_jvm" method returns a pointer to current JVM, which can be used for creating java objects and accessing their values.

CommOrg class is responsible for organising the communications between all other classes to accomplish a task given by a user. It has methods that start communication with one or more external devices, for example "comm_arduino_with_robot" method takes coordinates from a robot and sends them to arduino board. The "communicate" method is more advanced because it has access to all connected devices and it can decide what information goes where. By default when system is idling it just updates the map with available information, but when "Detector" or "JoystickIO" get new coordinates or "JIO" updates the robots position the method will try to update the states of all connected devices to make sure that they are up to date and can provide correct feedback if user is to need it. The communications can be started via "start_communication" method that takes a Boolean, which tells it if the code is started from java or from console and based on it the method creates a new thread for correct communication type and starts it.

The python code with DarkFlow and Yolo can be downloaded from GitLab [11].

2.4.4 Java code

There are 3 programms written in java GianControlCenter, GianStream and GianRobot.

GianControlCenter is a program for users, it provides a graphical user interface (GUI) that allows users to starts the system simply from the UI. It is also responsible for communicating with "GianStream" and "GianRobot". The former two are written for smartphones and robots to communicate with the system and the communication goes through the GianControlCenter. It is the top level program that manages everything.

There are two main packages in GianControlCenter "gui" and "logic". The first package contains all classes that make the graphical interface illustrated in Figure 2. The way code is designed allows different parts of the UI to be modified without affecting the looks or functionalities of other prarts. The "gui" package contains 9 classes 5 of, which are panels that contain different UI components. "GIANControlPanel" makes the top portion of the UI. The 3 radio buttons and the "Start GIAN" button are in this class. The "GIANGridPanel" creates the grid located at the left side of the UI. This is where user's hand position is displayed. The "GIANInfoPanel" is

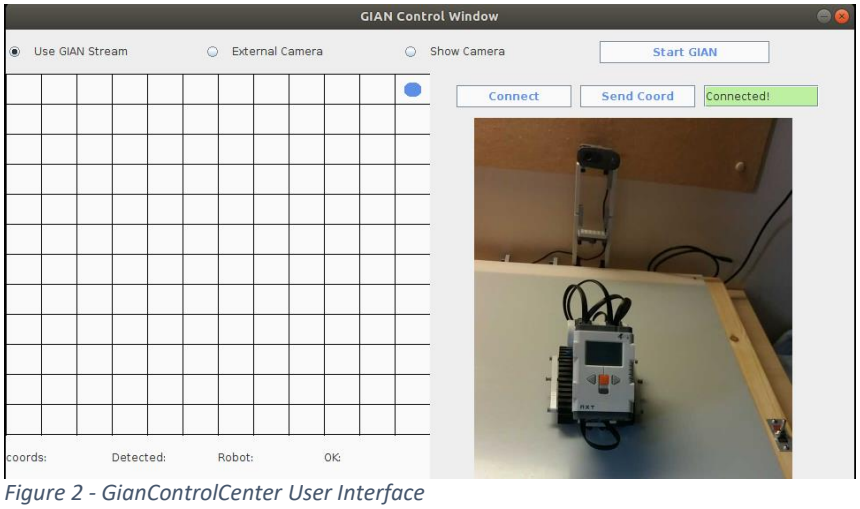


Figure 2 - GianControlCenter User Interface

located right under the grid, this is where more detailed information about hand and robot is displayed. The "GianRobotPanel" contains UI components for connecting and sending coordinates to robot. This panel is located on the right side under "GIANControlPanel".

The system has a feature that can connect to smartphones and receive images from them. These images are displayed using "Screen" class, which is also a panel. It uses "ImageIcon" class from "java.awt" package to do so. This panel is located under "GianRobotPanel".

The "Main_Window" class is responsible for taking all previously mentioned panels and putting them in the right places, thus creating the UI window. There is a "custom_objects" package in "gui" that contains two classes, one is used for creating custom designed buttons for this project and the other one is custom colours class with static RGB colours and colour gradients. These classes are used for decorating the panels.

The "**logic**" package contains 3 other packages "java_to_py", "main" and "tcp". The "java_to_py" package contains classes that manage the communication with python code. The "MyStack" class creates the stack we covered when describing the pythons "JIO" class in section (**2.4.3.1**). The "ProcessCreator" class allows java to call Linux shell commands and is mainly used for starting python scripts. The class has two "lunchScript" methods one that needs only a script location and one that can also have additional arguments for the script. We can use the second method to tell the python if it needs to use the "JIO" class or not by passing "withStack" argument to the script.

The "tcp" package contains two classes "RobotConnector" and "TCPIPServer". The last one is used for connecting to smartphones via tcp/ip socket and this class opens the socket.

The "RobotConnector" class uses "lejos" library to look for Bluetooth devices that are Lego bricks, once a unit is found the system will automatically connect to it. The class also have methods that provide information about connected robot including robots position, orientation and option. The last one can be anything that robot needs to communicate. For example it can tell if it's selected for an action or if it's ready to move. The "sendData" method sends coordinates to robot that are selected by gestures or by other input methods.

The "main" package contains two classes "Main_window_logic" and "StartGianStream".

The last one is responsible for opening a TCP/IP socket and listening to it for byte arrays on a new thread. On another thread if an array is received the program will try to create an image from it. This part can be modified to do more things for example it can look for other type of data and create different objects. To

achieve the communication it uses "TCPIPServer" class.

The "Main_window_logic" class is responsible for managing interactions between all other classes from lunching the python scripts to updating GUI and processing data received from robot. The class is divided into 5 sections "Robot methods", "Python communication", "GUI", "The Control" and "GianStreamPC". First, second and the last sections mainly send or receive data with according platform and by using supporting classes. The "GUI" section has a one method "updateSystemInfo" that collects data from the communicators and displays them on computer screen for users. It uses multiple methods from classes located in "gui" package. The "updateHandPos" method from "GianGridPanel" class is used for displaying user's hand position received from python. It also displays information received from joystick, smartphone and robot so that the user is aware of the states that the external devices are in at that moment.

The "The Control" section has a method "controlRobot", which takes data from communicators and based on them it decides if user wants to operate the robot or not, and if so what the user wants to do. When the final decision is made it is sent to robot for execution.

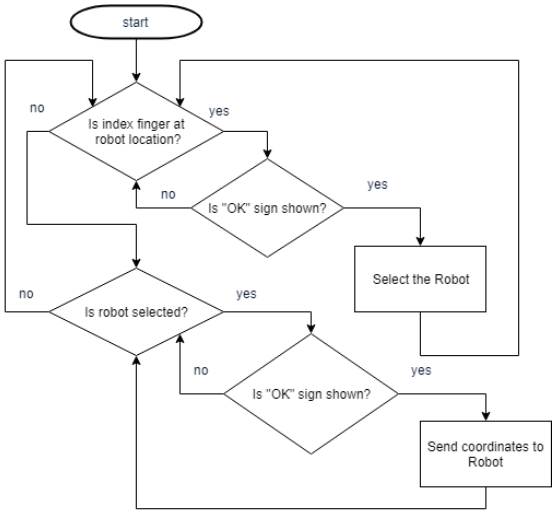


Figure 3 - Robot Selection Diagram

Since system can understand two gestures the logic of the method goes as shown in Figure 3.

The robot can be controlled from the UI too. There is a text box where user can add coordinates like "10,9" and when "Send Coord" button is pressed the coordinates will be sent to robot, which then will decide if it can go to that location or not. The method that does this is located in "Robot methods" section and is called "sendDataWhenPressed".

2.4.5 Android Code

GianStream is an android app that enables smartphones to connect and stream their camera footage to a given tcp/ip socket. The app is developed with android studio. The GUI of the app is programmed in xml and contains simple UI elements (Figure 4). It contains two text fields, one button and a TextureView object. The text fields are used for providing servers IP address and port number, which can be obtained from "GianControlCenter". When "connect" button is pressed the smartphone will start streaming its camera footage to a given server as well as displaying it on TextureView object. There are 4 classes that make this work "LunchCam", "CompareSizeByArea", "pcSocketClient" and "Main".

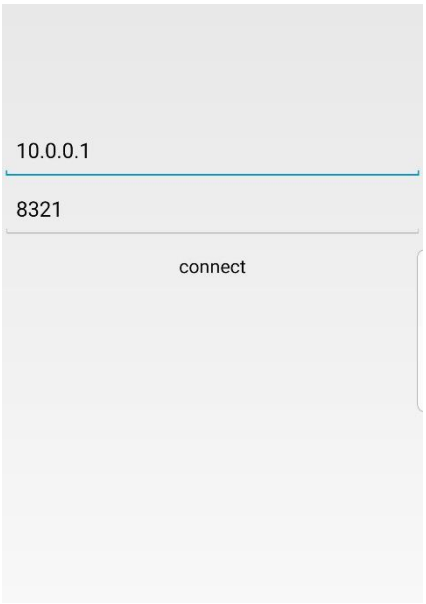


Figure 4 - GianStream GUI

"pcSocketClient" is used for connecting to a pc, its "makeClient" method is responsible for connecting to one. The IP address and port number have to be supplied to this method. The "streamBitmap" method is responsible for compressing images captured from camera and for sending them to pc.

The "LunchCam" class is using "Camera2API", which is a complex API, because it uses data pipelines and a lot of API methods have to be used for getting a raw footage from camera. Luckily google provides a sample code for how it is done and this class is heavily based on that sample [12]. The "startPreview" method of the class takes the "TextureView" object and draws the captured frame on it after resizing the frame to fit it on smartphone's screen. "chooseOptimalSize" method is used for resizing, which uses "CompareSizeByArea" class to compare the size of the image to smartphone's screen size.

The "Main" class is used for starting the app by creating necessary objects and managing the communication between them. The default "Gradle" solution generated by Android Studio is used for building the app.

2.4.6 Robot code

This code is responsible for controlling the robot's movements and keeping track of its location. It is divided into 4 sections "Bluetooth communicator", "Path Planner", "Path Navigator" and "Calibrator".

Bluetooth communicator section has "BTReceive" class that enables the Bluetooth communication and waits for incoming requests to connect. When communication is established it waits for incoming data that contains coordinates, after receiving the coordinates it replies with robot's position, orientation and options information, and then it goes to process the received data. If the data is different than the one it already had the old one gets replaced with the new one. The new data can be obtained from this class via "getCoord" method, which return the coordinates. The class has a "communicateWithPython" method, which attempts to communicate with python code directly. Even though the method works, still there are issues with maintaining the communication, which could be an incompatibility issue with python's "Serial" library. The communication with java is more reliable, thus it is the default communication method.

Path Planner section has "PathPlanner" class that is responsible for constructing a path for the robot to follow**.** There are many different techniques for robot path planning "A* search", "occupancy grid map" and "potential field map" to name a few. The system is already large and complex and to keep the robot code relatively simple a less complex and not very scalable method is used. For this to work we have to make couple of assumptions, we assume that the space where robot is going to be manoeuvred will never change, it will be 12x12 empty grid where robot can freely move in every direction. We restrict the robots ability to make turns at various degrees and limit it to do only 90 degree turns. Whit this assumptions in mind we create 2 methods "planFromNorthToSouth" and "planFromSouthToNorth" these methods are responsible for creating instructions that the robot is going to execute. We need two method because the robot's perspective changes depending on where it is located and according changes have to be made to the instructions. Both methods use dead reckoning localisation to estimate robots position and plan a path based on pre-programmed rules. The planned path is returned as a list of arrays that contain two values in them. The first value tells the robot what to do, 0 for rotating and 1 for moving. If first value is 0 than the second value tells the degree of the rotation, positive numbers for turning right negative numbers for turning left. If the first value is 1 and the second one is positive it means that the robot has to go forward for given amount, it will do the same but backwards if the second value is

negative. The "getPlannedPath" method is used for deciding on, which planning method to use and to do so the method takes 3 arguments, the robot's current position, the destination coordinates and the robot's orientation.

Path Navigator section has "PathNavigator" class that can take the instructions generated by path planner and drive the robot according to those instruction. The class has many methods that are responsible for different robot functionalities. The "followPath" method is the one that takes the instructions and goes through them to execute the commands, but to get the robot going it uses two other methods "rotate" and "moveForward". The former two make the robot turn or travel accordingly and they use "findRotationOffSet" and "findTravleOffSet" to correct robots on-board odometery errors. These methods use the formula $y = mx - \mathrm{b}$ where x is the amount travelled and y is the offset. To get this formula to work we have to find good values for m and b. To do that we have to take measurements by observing robots behaviour. The **Calibrator** class can be used for that. This class can find robot's wheel diameter and track width using the LED matrix, which are useful for setting up the "DifferentialPilot" class (*responsible for driving the robot*) from "lejos" library. The class also has a method "moveRobotWithIntervals" that makes the robot move forward for 2, 4, 6, 8, 10, 12 and 14 centimetres with intervals that are supplied to it in seconds. We can measure the offset of the robot for each distance manually. By doing so we will have y offset for every x distance and now we can use this information to estimate the values of m and b. To make this process easier a python script "linearfit.py" [13] was written, this script will produce the m and b values. To get it working we have to supply the x and y values to the script. The easiest way to do this is by opening the script and adding the numbers in corresponding lists on lines 10 and 11 (***x_list*** *for* ***x distance*** *value and* ***y_list*** *for* ***y offset*** *values*). Once we have the m and b values we can substitute them in the according formulas after, which our robot should be more accurate when navigating around.

Finally there is a **Main** class, which uses methods from previously mentioned classes to control the robot, for instance it will take new coordinates from Bluetooth class, will get robots current position and orientation from odometery and will use "getPlannedPath" method from "path planner" to get the instructions to supply those to "followPath" method from "path navigator" to get the robot going. It will go over this process again if the received coordinates change.

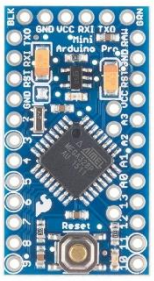


Figure 5 - Arduino Pro Mini

2.4.7 C/C++ code

There are 144 LEDs (light emitting diodes) that need to be turned on or off. These LEDs are arranged in a 12x12 grid and are used for providing feedback about what system is doing or is expecting user to do. Since using 144 microcontroller pins is not an efficient or practical way of controlling LEDs (*controllers with that many pins are very bulky and expensive*) a smarter solution had to be made. If we divide the LED grid into 4 parts we will get 4 sections each having 36 LEDs to control, this will simplify the complexity of the circuits.

There are many methods for controlling a lot of LEDs with relatively small number of pins. Charlieplexing and traditional Multiplexing are some of these methods. The first one is more effective in terms of pin saving, but small issues with an LED might cause the whole circuit to behave abnormally, furthermore building the circuit for Charlieplexing is not an easy task. More information about the issues with using

Charlieplexing can be found here [14]. Multiplexing is a better option, because the design is fairly straight forward and it is easier to code, but more pins have to be used. 36 LEDs can be placed in 6x6 grid, meaning we will have 6 negative and 6 positive ports, which means we will need 12 controller pins to individually control 36 LEDs, thus saving 24 pins. Arduino Pro Mini (Figure 5) boards, which are small and efficient have just enough pins to do this. Since there are 4 parts we will need 4 boards. Only 3 of the boards are going to be Arduino Pro Minis, the 4th one has to be an Arduino Uno. This decision was made because the programmer used for Pro Minis did not correctly work with computers serial communication, but Uno worked just fine. So with the current setup we have 1 master Arduino (Uno) and 3 so called slave boards (Arduino pro mini). The master Arduino receives commands from PC and broadcasts them to other Arduinos. The boards use I2C communication to talk to each other. This communication method was chosen, because it allow Arduinos to use their analogue pins for communication rather than digital ones, since those are used for controlling the LEDs. There are 4 Arduino script 3 of which are not very different, because they are only receiving coordinates that need to be lit up. The difference is the limits or the range of the coordinates the board has to react to. Those limits are set in the "loop" method in form of if statements. Each Arduino has different set of limits.

The master Arduino does the most computation, it decides what shape to show and which LEDs must be turned on for achieving that. All starts with it receiving an array from PC that contains 4 values. The first 2 are coordinates for an LED from, which shapes have to be extended, in other words if there is a shape its (0, 0) coordinate has to be the arrays first two values. The third value tells the orientation of the shape it can be a number between 1 and 4, which will correspond to North, South, West or East. The 4th one is the shape option, for example "0" means show finger shape or "non 0" for robot shapes and "4" is for robot shape when robot is not at the location. When robot is selected we will use the command "4" to show the robot shape as a finger location so that user knows that they have selected the robot. Every shape and orientation is programmed in a method of its own, so for example "showFingerLoc" turns 4 LEDs on depending on where the user is pointing. "showRobotEast" method will turn on LEDs to display robots shape facing east and so on. There are 4 additional shapes "G","I","A" and "N" these letters are displayed one after another to greed the user. The matrix will displays the initial location of robot so the user know where system things the robot is when the system starts.

When looking into these methods it might get a bit confusing and hard to understand what they are doing, because of all "x" and "y" values and coordinates, but everything is quite simple. Every number added or subtracted from "x" and "y" values just tells that we want to go that many LEDs away from the current location and turn them on.

2.5 Hardware

2.5.1 LED Matrix

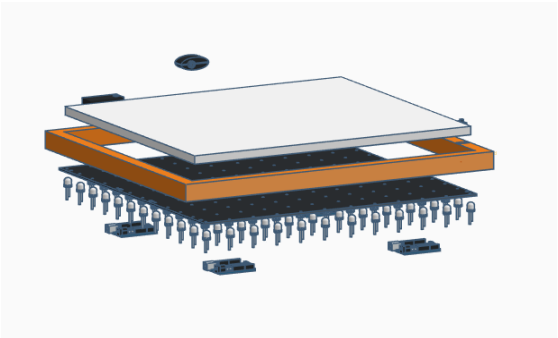


Figure 6 - 3D design of the LED matrix

As mentioned before the LED matrix is controlled by 4 Arduinos. Since there are 144 LEDs each Arduino has to control 36 of them, this means that each Arduino pin has to be able to turn on 6 LED with their max brightness at the same time. According to the datasheet [15] of the LEDs used in this project the current draw per LED can be about 30mA. The 6 LEDs

will draw about 180mA, this means that every pin of the Arduino has to supply 180mA current, however from Arduino website [16] under “Tech Specs” section we can find that Arduino is only capable of supplying 40mA current per pin. This is clearly not enough and given that we have 36 LED we will need a solution that can provide around 1A current. Obviously we have to use an external power supply and to deliver the external power to LEDs we need circuit capable of delivering this power.

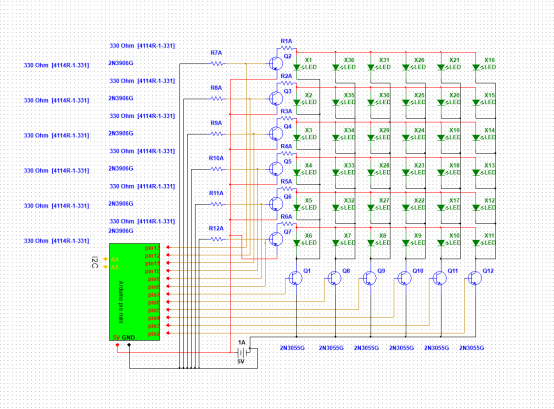


Figure 7 - 6x6 Matrix circuit schematic

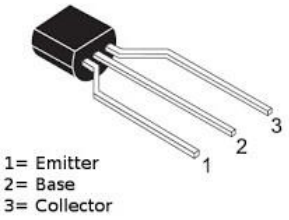


Figure 8 - Transistor

On Figure 7 we can see the circuit design that is going to be used. It consists of six PNP and six NPN transistors and twelve 330 ohm resistors. The LEDs are placed in a grid that produces 6 columns and 6 rows. The cathodes (positive side) of the LEDs are connected to the columns (6 LEDs per column) and anodes (negative side) are connected to rows (6 LEDs per row).

Figure 9 – physical 6x6 Matrix circuit board

PNP transistors are used for connecting positive side of the power supply to the columns. PNP transistor lets the power through when its base is grounded and restricts it at positive voltages. The Arduino boards can’t turn their pins into grounds (GND), so to create a negative potential at the base we need to connect it to the ground physically. We need to add a resistor to the base after connecting Arduino pin to it to prevent short circuit when Arduino pins supply positive potential to transistor’s base. Another resistor has to be added to the collector pin (pin that the column is connected to) of the transistor to protect LEDs from accidental current spikes. The positive side of the power supply is connected to the emitter of the transistor.

The NPN transistor is activated when the base pin has positive potential and is deactivated when it has negative one. No resistor is needed for this transistor because no danger of damaging the circuit is present since a resistor is already connected to LED row. We mentioned before that Arduinos can’t produce negative potential, and it will not be a problem, because this transistors are activated at certain threshold, which means if we supply positive potential at a level that is below the threshold the transistors will be disabled. On the second page of the transistors datasheet [17] we can see that it activates at potentials above 0.4 volts.

This circuit design first was tested on a breadboard for 3x3 matrix, which performed as expected [18]. After successfully testing the circuit an efficient method had to be found for making 4 of them for each of the 4 6x6 matrices. Aligning 36 LED correctly for soldering wires to them can be very challenging if is not done well. To simplify the process 3D models were created. This models allow LEDs to be easily aligned by having designated spots for them, which makes the soldering process bit easier and less time consuming. The 3D models were created by Tinkercad [19] online 3d designing software

and printed with Wanhao D6 [20] printer. Total of 16 pieces were printed.

The 3d models can be found in project GitLab.

Figure 10 - printed 3D model piece

There are two configurations per model we need two copies of both. When all pieces are printed they can be enforced together with nuts and bolts. To keep the printing times small the models were printed with fewer layers, which made them fragile, so to keep the models and the circuit safe a case was needed. The case was also designed with Tinkercad, but it was made from wood. To completely eliminate 3d models contact with external objects the frame was covered with Acrylic glass. The frame also houses a switch that can be used for turning the matrix on or off and it has structures where external power source and a camera can be placed. The button on the back of the frame can be used for resetting the boards.

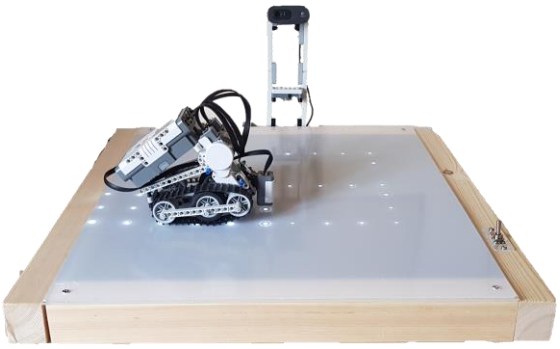


Figure 11 - Final version of the LED matrix

2.5.2 Robot design

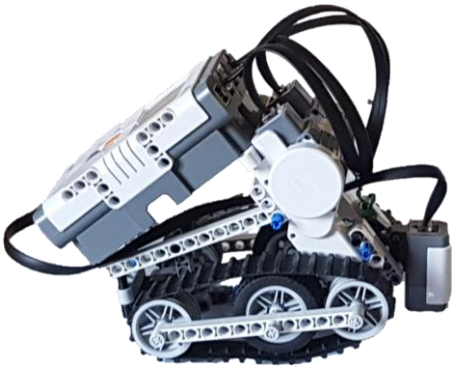


Figure 12 - The Robot

The robot is built with "LEGO Mindstorms NXT" kit. It has two motors and a light sensor. The sensor is used for calibration we discussed in section 2.4.6. The design of the robot is not very important, because the system does not care what the robot looks like as long as it is calibrated and can plan its navigation path properly. The robot used for demonstrations has 3 wheel on each side for stability and the LEGO brick is on top for easy access see Figure 12.

2.6 Control of the system

The compiled versions of the developed programs with the exception of python can be obtained from Gitlab in precompiled form, hopefully making the installation process easier. After successfully installing all necessary components the system should run without problems.

Before the system can be started all external devices (LED matrix, Camera and optionally Joystick) have to be connected to the computer. Next the "arduinopermitions.sh" file has to be executed this will allow Linux to communicate with Arduino. Then we have to make sure that the "GIAN.sh" file is copied to "/usr/GIAN" folder. Once all previous steps are completed the "GianControlCenter" can be lunched. Next the robot needs to be turned on and the default program should be activated. To connect to the robot the "connect" button in "GianControlCenter" should be pressed. When robot is connected the "External camera" and "Show Camera" options can be selected and "Start GIAN" button can be pressed. If everything is done correctly a window will pop up with two options "Detector" and "Joy". If user wants to use the gestures the "Detector" option should be selected, the other option will start the system using the joystick.

Now user can point at the robot and wait for LED matrix to highlight the area where robot is located, then an "OK" sign can be shown. If the system recognises the sign the robot will make a "beeping" sound indicating that it is selected.

After that user can point to other location and if the location gets highlighted with the robot's shape than user can show the "OK" sign again to move the robot there. When robot moves to new location the process can be repeated.

3 What is next

The system can be expanded to support more devices that can add extra gestures and other input methods to it. For example a smart watch can be used for determining the hand position so that the camera can be used for detecting the gestures only or for doing "Spontaneous spatial coupling" [21], a technology that can turn every object into controller, making the system more usable in large spaces.

Finally upcoming 5G network can be used for connecting all devices to the system, which has the potential of improving the communication speeds and reliability of the system.

4 Project planning

It was quite a journey from an idea to building a working product. A crucial research was taken during the summer of 2018 to understand and find technologies that can help bring the idea into reality. During this process many different approaches were considered for each part of the project. For example it was decided to use a foam for aligning the LEDs for the matrix, but after discovering the availability of a 3D printer this decision was changed in favour of the printer and the reason was the precision that we could achieve with it. Also "closed palm" gesture was considered for selecting the objects, however later in development stage I realise that the gesture is very similar to the "index finger" gesture, which could cause confusions to neural network leading to wrong predictions. The solution was to replace is with "OK" gesture.

Version control and project planning software were used for all stages of the project.

Taiga (project management software) was used during summer research, but latter during the development "Jira" and "GitLab" were used.

Agile development methodology was used for project management, more specifically Kanban.

The Jira software allows us to create different types of issues (Story, Epic, Task, Risk, Bug and feedback) to populate the Kanban board. All issue types were used for managing this project except "Story". The stories are used when multiple developers are working on the same project and they need to know the context (or scenario) at, which their code or product is going to be used. Since only one person is developing the system there is no need for user stories.

The project was divided into multiple epics and each epic had multiple tasks and later in development cycle Bugs and ricks associated with it. When all tasks of an epic were completed the epic was considered done and was moved to appropriate column on the board. Figure 13 illustrates the quantity and types of issues used in this project.

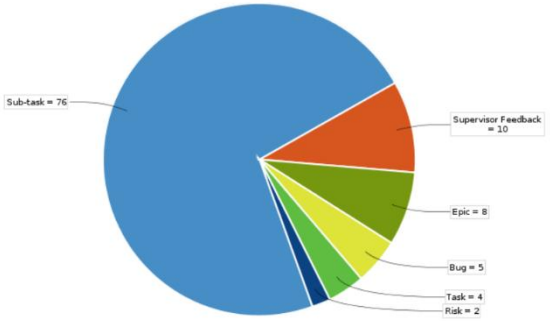


Figure 13 – Pie Chart: Used JIRA issue types

From the chart we can see that total of 8 Epics were used. The Epics were for JAVA code, Python code, c/c++ code, LED Matrix, Robot and android app. The other two Epics were used for updating GitLab documentation and for working on the report. We can also see that there were 76 sub-tasks this is the total number of tasks created for all Epics. It is worth noting that two Epics are still active, meaning that the number of sub-task might increase over time. There were 2 risks taken for

optimising the gesture detection code. Those were risks because a lot of time was needed for completing the optimisation and the results might have not been significant, but fortunately it paid off with significant improvements in performance.

There were regular meetings with project supervisor to discuss the progress of the project and in addition to this supervisor feedbacks were added to the Kanban board. If any suggestions were made in those feedbacks, according changes or modifications were made to test them and the results were shared with the supervisor.

GitLab was used for version control. The developed designs, files and code were stored in there along with technical documentation to make sure that safe backup of developed content is available if it is to be needed. Figure 14 shows the GitLab activity throughout the system development. From the graph we can see that there were active contributions and updates during the development period with total of 154 commits to the repository.

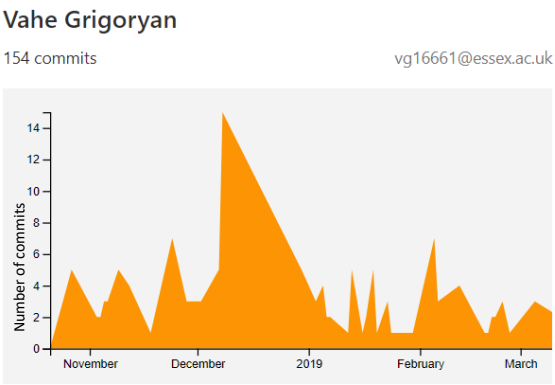


Figure 14 - Git commit history

5 System Testing

The developed system can be used for operating a robot on a constant environment, but many issues are still present, which can be frustrating for someone who does not know how it works and where the problem might be coming from. A care was taken to make the system as user friendly as possible in a given amount of time. For example a graphical user interface was developed to make the interactions with the system easier and joystick was added to the system for users to familiarise themselves with the system using more traditional input method before using hand gestures. Even with the improvements some individuals did not find the system very intuitive, because there was a lag forcing user to wait for system to catch up, but mainly positive reactions were received when joystick was used.

The system has many parts that communicate with each other, so the lagging could potentially be caused by any of them. To figure out the source of the problem we had to reproduce the issue using debugging mode on java and python. Unfortunately reproducing the problem when the codes are run separately was not possible so a different method had to be used. A code was added to python to produce files that contain debugging information. Now we can run system the usual way while getting debugging information about java and python at the same time. It appeared that the problem was caused by "py4j" library, but further investigation showed that the GPU might be causing the real issue. The GPU of the computer that was used for the demonstration had only 2 GB of RAM, which was quickly flooded with data and when system started introducing unnecessary delays it interrupting python to java communication leading to errors with "py4j".

The lag issue can be solved if faster computer with better GPU is used or the CNN model can be stored on cloud for better performance, but this will mean that the system will have to communicate with a database, which can become an ethical issue.

Overall the system is functional, but many improvements to the speed and scalability have to be made before it can become truly useful in large environments.

6 Conclusions

With the expectations of the project fulfilled to a significant quantity, it will be safe to conclude

that all desired features of the system were implemented successfully with available resources.

The Idea was to build a system that is not connected to the internet and can control IoT devices with hand gestures. The developed product can do everything that was planned and it also supports an additional input method, a joystick, giving users a choice between traditional and gesture based controls.

A matrix containing 144 LEDs was built, which is controlled with 4 arduino boards. The boards communicate with PC where Python code sends them commands based on Convolutional Neural Network's predictions. The predications are also shared with Java code, which displays them on computer screen via GUI and sends user commands to a robot. The robot plans a path and navigates it on the LED matrix based on received data, which reflects what user wanted to do with the robot.

In the report we described the technologies used in each part of the system, while justifying the choices, we discussed agile project planning that was used throughout the project and finally we demonstrated the extent to what the system was able to deliver the promised result with system testing.

All developed files and more details about the project can be found in the GitLab [9] repository.

7 References

[1] F. Paul, “Why IoT for seniors is a lot tougher than it looks,” [Online]. Available: http://bit.ly/2KdSxhj. [Accessed 31 03 2019].

[2] “IFTTT website,” [Online]. Available: https://ifttt.com/. [Accessed 14 04 2019].

[3] A. Banafa, “Major Challenges Facing IoT,” IEEE.org, [Online]. Available: bit.ly/2GwGfN9. [Accessed 10 04 2019].

[4] V. Grigoryan, “Initial Report,” [Online]. Available: bit.ly/2GBPxr9. [Accessed 10 04 2019].

[5] “CUDA download Page,” NVIDIA, [Online]. Available: https://developer.nvidia.com/cuda-90-download-archive. [Accessed 04 03 2019].

[6] “Cudnn download Page,” NVIDIA, [Online]. Available: https://developer.nvidia.com/cudnn. [Accessed 04 03 2019].

[7] Trieu, “DarkFlow-GitHub,” GOOGLE, [Online]. Available: https://github.com/thtrieu/darkflow. [Accessed 04 03 2019].

[8] J. Redmon, “YOLO v2,” [Online]. Available: https://pjreddie.com/darknet/yolov2/. [Accessed 04 03 2019].

[9] V. Grigoryan, “GIAN-GitLab,” [Online]. Available: https://cseegit.essex.ac.uk/ce301/grigoryan_v/capstone_project. [Accessed 04 04 2019].

[10] T. Lin, “LabelImg-Github,” [Online]. Available: https://github.com/tzutalin/labelImg. [Accessed 06 04 2019].

[11] V. Grigoryan, “Python-GitLab,” [Online]. Available: http://bit.ly/2I1V4Jw. [Accessed 05 04 2019].

[12] “Camera2Basic - API Sample,” Google, [Online]. Available: https://github.com/googlesamples/android-Camera2Basic. [Accessed 18 04 2019].

[13] V. Grigoryan, “linearfit.py-GitLab,” [Online]. Available: http://bit.ly/2UbjGRw. [Accessed 10 04 2019].

[14] “Charlieplexing vs. tradutional multiplexing,” IPFS, [Online]. Available: bit.ly/2Zloh7u. [Accessed 10 04 2019].

[15] “Technical Data Sheet White LED,” Everlight, [Online]. Available: bit.ly/2UCNV90. [Accessed 14 04 2019].

[16] “Arduino pro mini Specs,” Arduino, [Online]. Available: https://store.arduino.cc/arduino-pro-mini . [Accessed 14 04 2019].

[17] “S8050 NPN transistor datasheet,” UTC, [Online]. Available: http://media.nkcelectronics.com/datasheet/s8050.pdf. [Accessed 14 04 2019].

[18] V. Grigoryan, “3x3 LED matrix test,” [Online]. Available: bit.ly/2KEo8sw. [Accessed 14 04 2019].

[19] “Tinkercad,” AUTODESK, [Online]. Available: https://www.tinkercad.com/. [Accessed 14 04 2019].

[20] “Wanhao D6 (3D printer),” WANHAO, [Online]. Available: bit.ly/2PahXeg. [Accessed 14 04 2019].

[21] C. Clarke and H. Gellersen, “Gesture control tech turns any object into a TV remote,” Lanchester University, [Online]. Available: bit.ly/2GsQcem. [Accessed 16 04 2019].

8 Appendix

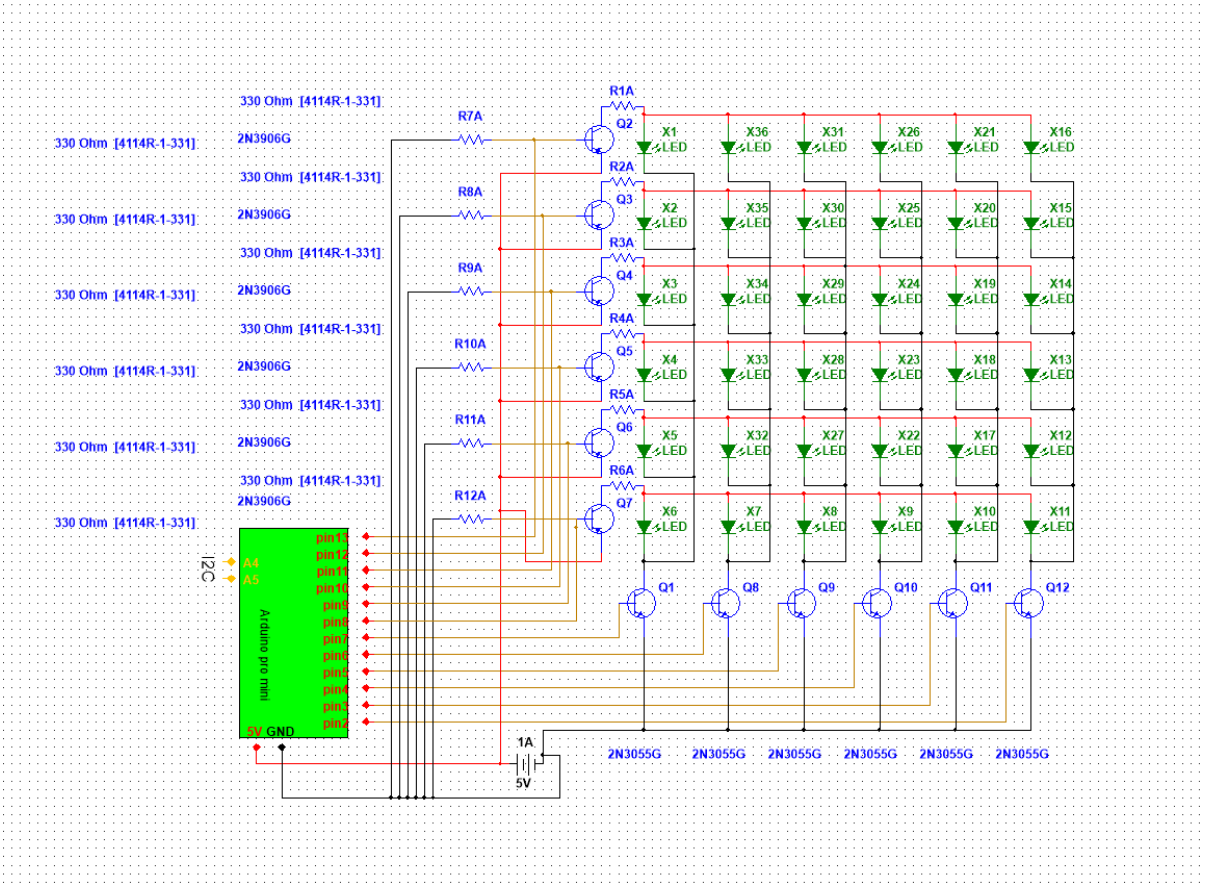


Figure 15 - 6x6 LED Matrix circuit schematic