

**Bachelor of Electrical and Electronics Engineering with Honors**

**EEE4336 INDUSTRIAL TRAINING**

**FINAL REPORT**

|  |  |
| --- | --- |
| **Name** | ZHANG Fengchu |
| **Student ID** | SUKD1801575 |
| **Organization** | Cheng Hua Engineering Works Sdn. Bhd. |
| **Industry Supervisor** | Mr. TAN Chung Men |
| **Academic Supervisor** | Mr. Chris CHONG Hock Siong |
| **Duration** | 2021-06-01 - 2021-08-20 |
| **Date of Submission** | 2021-10-15 |

# Abstract

This report details the task performed and experience gained as an intern at Cheng Hua Engineering Works Sdn. Bhd. from 2021-06-01 to 2021-08-20 as a part of the third year Industrial Training course (EEE4336).

Overall this report will explain in detail the background of the company, the description of all the tasks and projects performed, and finally an analysis of the problems that occurred during the internship both in the technical, and management aspects.

The internship was divided into three phases:

Phase 1: Understand warehouse operations, and system architectures.

Phase 2: Feasibility study, and researching existing AGV systems.

Phase 3: Product development.

It will mainly focus on Phase 3, where an attempt was made to develop electronics and software systems for a QR code guided Automated Guided Vehicle (AGV) fleet for the purpose of carrying a 5 kg load in a 3 meter high warehouse designed to store e-commerce goods. This part introduces the industrial context which necessitates this development, followed by the process by which the design concept came to being, followed by the testing and prototyping stages, then the critical analysis of the problems faced during these stages. Also highlighted are the the administrative aspects of the product development such as cost management, time management, role delegation, and potential future work that can be done on top of what was left off.

It must be noted that due to time constraints the full prototype is yet to be completed. The focus thus would be on the camera QR detection, communication between AGV and server, and motor control.

Since I was working with another intern, Sara Hany Tawfik Hussien, when doing this project, the aspects of the work I completed will be the focus while referencing S.. Hussien’s work if needed.

# Acknowledgement

Firstly, I sincerely express my gratitude to Mr. Chris Chong Hock Siong for going through the extra effort to personally help us find this internship opportunity at a time when, due to the pandemic, many students including us were unsuccessful in our endeavor to find a placement on our own. Also great thanks to your continued support as our academic supervisor during and after the internship.

I would also like to give special thanks to Mr. Tan Chung Men, industrial supervisor, for being a patient, big brother figure for us, and an excellent guide to the real world of industrial product design. Without you, our tasks surely wouldn’t have been completed.

Next, I would like to thank my co-intern Sara Hussien, who was so much more than a wonderful teammate, and an inspiration for me to always keep striving to hone my discipline and skills.

Finally, I would like to thank Electrical Engineering department of SEGi University for providing us with this course; and Product Development Manager Mr. Sam, HR and Admin Asst. Manager Ms. Penny Tan, Mr. Yew, Mr. Gan et al. from Cheng Hua.

# Disclaimer

According to the Nondisclosure Agreement the interns have with the company, through Mr. Sam, all videos and codes made by us during the internship are to be considered company intellectual property and cannot be freely shared with the public.

We have been given permission to include them in this report.

# Table of Content

[Abstract ii](#_Toc19563)

[Acknowledgement iii](#_Toc27629)

[Disclaimer iii](#_Toc14013)

[Table of Content iv](#_Toc2171)

[List of Figures iv](#_Toc12836)

[List of Tables v](#_Toc30763)

[Chapter 1: Introduction 1](#_Toc15868)

[Chapter 2: Company Background 3](#_Toc16904)

[Basic Information 3](#_Toc7639)

[Services Provided by the Company 3](#_Toc22852)

[Corporate Structure 3](#_Toc26207)

[Chapter 3: Description of Tasks and Project 4](#_Toc15643)

[3.1 Overall Concept Development 4](#_Toc18993)

[3.1.a Design Criteria and Context 4](#_Toc3261)

[3.1.b Concept Development Process 4](#_Toc28322)

[3.2 My Share of the Work 8](#_Toc22407)

[3.2.a D\* lite Algorithm 8](#_Toc3979)

[3.2.b Motor Control, Differential Steering, and Object Detection 8](#_Toc29721)

[s − length of the route covered by the wheel 14](#_Toc12940)

[3.2.c Critical Analysis 20](#_Toc15629)

[3.3 Costs 21](#_Toc28015)

[3.4 Project Management and Workflow 21](#_Toc18511)

[Chapter 4: Problems Occurred and Description 22](#_Toc8330)

[4.1 What could have been be the next steps? 22](#_Toc5748)

[4.2 What would we have done if resources weren’t limited? 22](#_Toc19799)

[Conclusion 22](#_Toc24955)

[References 23](#_Toc12951)

[Appendix 24](#_Toc11995)

[Appendix A: Meeting Minute Summaries 24](#_Toc25271)

[Appendix B: Codes 28](#_Toc9940)

[MDD10A.py 28](#_Toc28254)

[AGV\_MOVEMENT.py 31](#_Toc847)

[Appendix C: Data Sheets 41](#_Toc19518)

# List of Figures

Figure 1: Pg 5

Figure 2: Pg 9

Figure 3: Pg 10

Figure 4: Pg 11

Figure 5: Pg 12

Figure 6: Pg 13

Figure 7: Pg 20

Figure 8: Pg 20

# List of Tables

Table 1: Pg 21

Table 2: Pg 22

# Chapter 1: Introduction

As industrialization and automation further entrenches itself within all aspects of society, it is essential for any business to improve its performance by upgrading technology in terms of robotics, information management, and automation. This is especially true in industries traditionally reliant on manual labor, in order to reduce costs, improve workplace efficiency, improve safety, and ensure feedback data collection for further improvement.

In the warehouse management industry it is no different. In the previous decades, various types of warehouses have implemented various types of robots, such as AGVs, AMRs, etc. to simplify and automate the industry practices of put away, replenishment, storage, picking and so on, each with its own specialty. Large e-commerce companies such as Alibaba and Amazon are examples where the implementation have yielded great results, thus encouraging middle and smaller businesses to innovate in these fields, and produce machines that join the competition and aim to be adopted by other e-commerce companies that would like to repeat this success.

As interns at Cheng Hua Engineering Works, Product Development Department, our main task was to aid in the early development of an AGV system that could, at the very minimum, have a fleet of AGVs that could each carry a load of 5kg, move in a 4-way grid fashion, and automatically navigate to any desired location on a single floor of a warehouse, without any collision, based on detecting coordinates encoded in QR codes pasted on the ground.

By the time this report was written, due to constraints that will be discussed later, preliminary work has been done on the QR detection, communication between AGV and server,motor control, object avoidance, and navigation algorithm.

In terms of timeline of events, the duration of the internship was divided into three phases, where the Phase 1, completed within a week, was to acquire basic knowledge of warehouse operation and software warehouse management systems. Phase 2, which lasted for one and a half weeks, was doing preliminary research on available AGV’s already on the market and how they could inspire our product development.

Both phases can be seen as background knowledge research for Phase 3, product development, and in this report the knowledge acquired in the first two phases will be “blended in” with the rest of the report rather than having their own sections.

# Chapter 2: Company Background

## Basic Information

* Full name: Cheng Hua Engineering Works Sdn. Bhd.
* Establishment year: 1967
* Address: Lot 10383/10384, Jalan Laksamana 20, Off, Jalan Sungai Jati, Taman Sentosa, 41200 Klang, Selangor

## Services Provided by the Company

According to the company website, Cheng Hua is a supplier of integrated material handling technology that serves the industries of logistics, tire handling, and automotive. It’s solutions include sortation machinery such as the megasort series, conveyors systems such as roller conveyors, palletizing systems such as chain conveyors, and vertical transfer systems.

The MegaBot product line belongs under sortation systems, however the robot itself is designed and made by Zhejiang Libiao Robotics Co., Ltd. Cheng Hua Engineering works wishes to design and build such a sortation robot in-house, and our internship was part of that process.

## Corporate Structure

Overall the headquarter of the company is in Malaysia, where it was founded. It has since expanded its branches to Shanghai, Bangkok, Chennai, and Tangerang (Indonesia).

The internal corporate organizational structure was not made available to us interns. However, the structure in relation to our internship was simple and as follows:

* HR and Admin Asst. Manager Ms. Penny Tan was responsible as the liaison between our university and the company.
* Product Development Manager Mr. Sam is responsible for overseeing all product development, and thus oversaw the operations of our internship.
  + PLC Engineer Mr. Tan Chung Men, our industrial supervisor, was directly responsible for us interns in all practical operations.
    - Interns

# Chapter 3: Description of Tasks and Project

## 3.1 Overall Concept Development

### 3.1.a Design Criteria and Context

#### 3.1.a.i Performance

The AGV was expected to perform as the following:

* Be able to carry 5kg load.
* Move along a 4-way grid, either following a line or not.
* Locate itself my scanning a QR code fixed to the floor, which contains information of coordinates.
* Upload coordinate location to server for further path planning.
* Automatically rotate or move forward at the correct amount when receiving commands from the server.
* Avoid unexpected obstacles.

#### 3.1.a.ii Safety

For safety, the primary concern that was to be solved in term of electronics, was for the AGV to be able to detect any unanticipated object within its vicinity, especially in its forward direction of travel, pause processes to stop the movement immediately, and avoid collision. The distance of sensing must be of sufficient length but not oversensitive, and the information during the pause needs to be relayed to the control server in order to theoretically update other AGVs in the same fleet of this disturbance and reroute around the paused AGV.

### 3.1.b Concept Development Process

The concept development process began by observing existing QR following AGV models, both professional and amateur, and inferring the underlying elements within.

After some preliminary research, the following top block diagram was created:

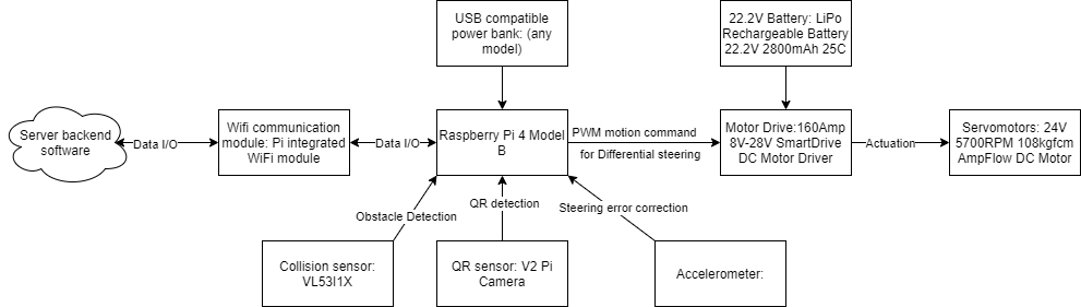


Figure 1: First draft Top level block diagram

Based on this design the following hardware components were considered:

Hardware components

Microcontroller/Single board computer​

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ​  ​ | ESP32​ | Raspberry Pi 4​ | Arduino Mega​ | Atlas 200 ​ |
|  |  |  |  |  |
| Processor​ | Single or Dual-Core 32-bit LX6 Microprocessor with clock frequency up to 240 MHz.​ | Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz​ | ​  ​  ​  ATmega2560​ | ​  ​  ​  Ascend 310​ |
| Memory​ | 520 KB of SRAM, 448 KB of ROM and 16 KB of RTC SRAM.​ | 2GB, 4GB or 8GB LPDDR4-3200 SDRAM.​ | 256 KB of which 8 KB used by bootloader. SRAM 8 KB. EEPROM 4 KB.​ | LPDDR4X, 8 GB/4 GB, total bandwidth 51.2 GB/s​ |

Camera

|  |  |  |  |
| --- | --- | --- | --- |
| ​ | Raspberry Pi camera V2​ | ESP32-CAM​ | Mercury series camera ​ |
| Resolution​ | 8 Megapixels​ | 2 Megapixels​ | 8.0 Megapixel ​ |
| Video modes​ | 1080p30, 720p60 and 640 × 480p60/90​ | 720P 15 up to 60 FPS​ | 1080p 60, 720p120 fps​ |
| Sensor​ | Sony IMX219​ | OV2640 ​ | Optical CMOS​ |
| Sensor resolution​ | 3280 × 2464 pixels​ | 1622×1200 pixels​ | 3280×2464 pixels​ |

Obstacles avoidance​

|  |  |
| --- | --- |
| Ultrasonic sensor​ | Distance Sensor VL53l1X​  ​ |
| An instrument that measures the distance to an object using ultrasonic sound waves.​  ​  Specifications:​  ​  Measuring Distance: 2cm to 80cm.​  Accuracy: 3mm.​  Operating Frequency: 40Hz.​ | A tiny, self-contained lidar system featuring an integrated 940 nm Class 1 laser, which is invisible and eye-safe.​  Fast and accurate ranging with three distance mode options:​  Short: up to ~130 cm, 50 Hz max sampling rate; this mode is the most immune to interference from ambient light.​  Medium: up to ~300 cm in the dark, 30 Hz max sampling rate.​  Long: up to 400 cm in the dark, 30 Hz max sampling rate.​  ​ |

The hardware selection criteria was as follows:

* Prices given without inquiry.​
* Available for quick and cheap shipment to Malaysia.  ​
* Standardized connection interface.  ​
* Datasheets and online resources readily available.

The final selection of components, see section “3.3 Costs”. Out of these, unfortunately the accelerometer and VL53L1X Ranging Sensor were not used in the prototype due to time constraints.

Since after this point, the tasks for each element in the block diagram was specialized between Sara Hany and Zhang Fengchu, the next two sections will be based on what each of us have done.

## 3.2 My Share of the Work

### 3.2.a D\* lite Algorithm

ZHANG only performed very preliminary research on D\* lite, and was unable to program the algorithm into python code due to the lack of knowledge in this area. The task was eventually delegated to the industrial supervisor Mr. Tan Chung Men, who designed and holds the python code.

An introduction and example to the decision mathematical theory behind the algorithm can be seen in this video by MIT:

<https://youtu.be/_4u9W1xOuts?t=2783>

In short, the D\* Lite path planning algorithm is an incremental heuristic search algorithm that can be applied to an entire fleet of AGV to manage navigation and avoidance of collision between them. This in turn is implemented in a server which communicates information with individual AGV units through and Internet of Things type network.

### 3.2.b Motor Control, Differential Steering, and Object Detection

The goal of the motor control is to use the Raspberry Pi to control the MDD10A motor driver, which in turn relays electrical power from the 22.2 battery to the 2 motors respectively both in term of rotation angular velocity and direction of rotation.

The main inspiration for both the circuit configuration and the python code can be found on <https://www.robotshop.com/community/forum/t/review-for-motor-driver-mdd10a-controlling-by-raspberry-pi/31740>, written by [Mohammed Omar Salameh](https://www.youtube.com/channel/UCubYb4lIdrh09Vme9GFtvCA).

The secondary goal is to use various types of sensors (Infrared and Ultrasonic sensors used here) to pause all movements of the motor when an object is detected.

#### Circuit Configuration

For a reference on the position of GPIO pins see <https://projects.raspberrypi.org/en/projects/physical-computing/1> (compatible with all Model B type Raspberry Pi).

##### Motor Driver, Motor and Battery

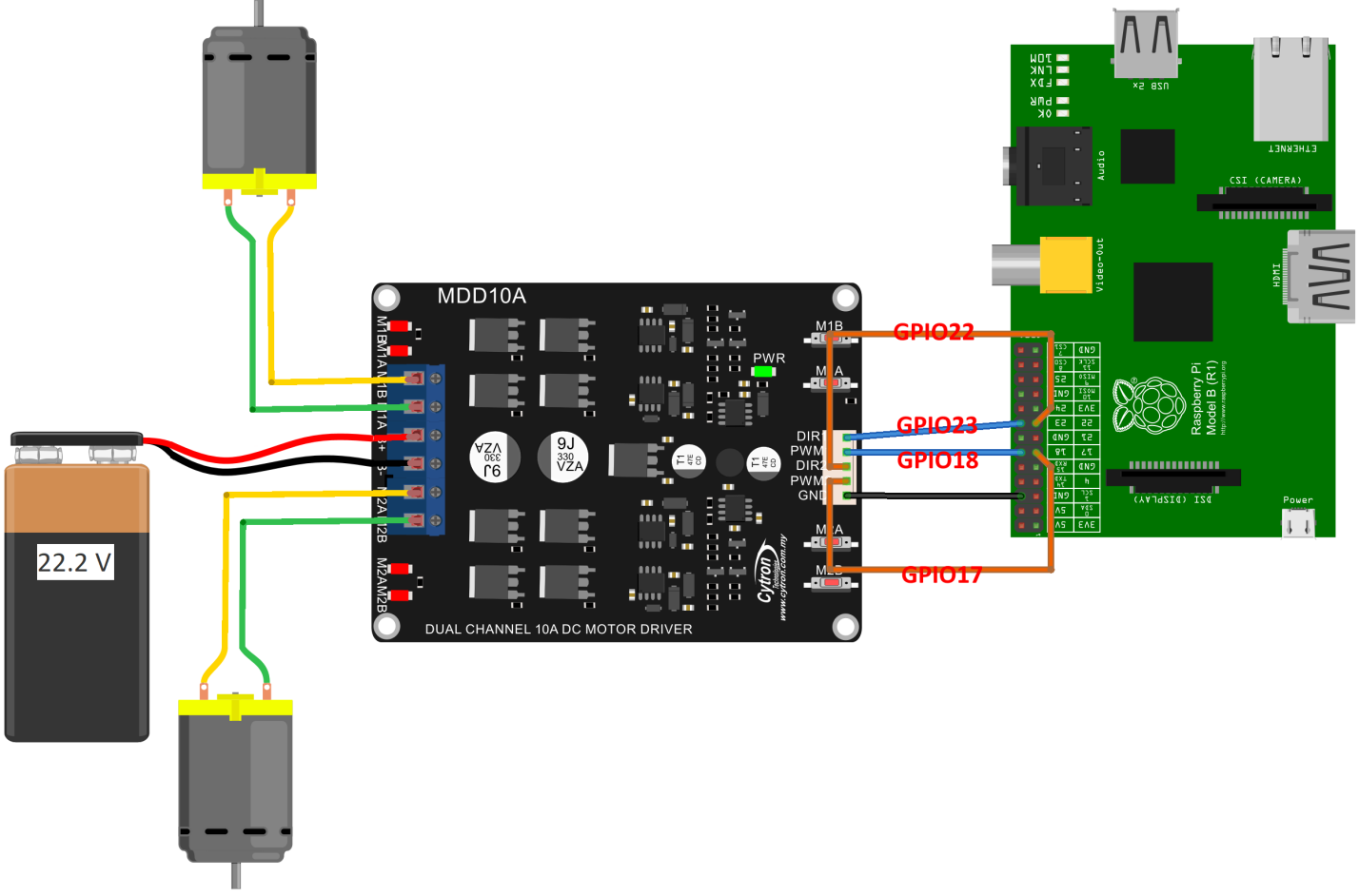


Figure 2: The circuit configuration for the motor control and differential steering module

The circuit configuration is as the figure above. Not shown is the power cable between the Raspberry Pi and a power source, and HDMI cable between the Pi and a monitor. Unlike the above figure, since Raspberry Pi 4 Model B was used during testing and prototyping, the power cable is USB-C rather than USB-B micro as depicted; and the HDMI connection is Micro HDMI instead of the standard port.

The Pi can powered just like a smartphone, where the USB cable is connected to a USB power brick which is in turn connected to a wall powerpoint; or connected to a power bank (for the vehicle to be “wireless”).

Also of note is that the GPIO pins do not necessarily have to be 17, 18, 22, and 23, as it is simply a matter of choice. Such configuration can be changed in the python code.

##### Object Detection Sensors

**Infrared Sensor**

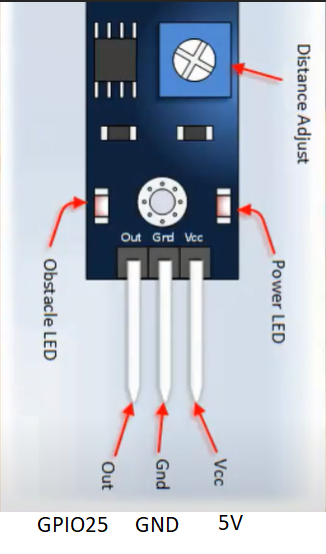


Figure 3: Connections of the IR sensor to the Pi

To be noted that in the above figure, again the GPIO can be any one of the many on the Pi.

**UltraSonic Sensor**

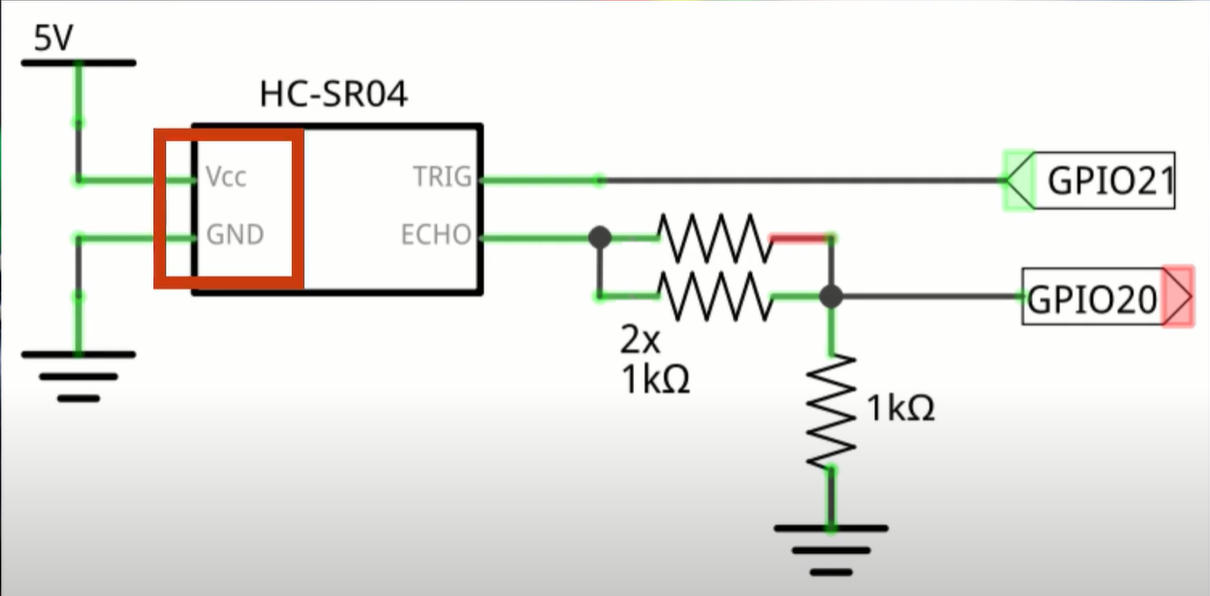


Figure 4: Connection of the Ultrasonic sensor to the Pi.

Since for the Pi GPIO rated voltage is 3.3V, while the output from the ECHO pin is 5V, a voltage divider is needed to reduce the input voltage, based on the following formula (note that the resistor values do not necessarily need to be 1 kΩ, as long as the ratio is correct, since the input impedance of the Pi GPIO can be assumed to be infinite):

Also to be noted here is the GPIO could be any on the Pi.

#### Theory: Pulse Width Modulation

Pulse-width modulation (PWM), or pulse-duration modulation (PDM), is a method of reducing the average power delivered by an electrical signal, by effectively chopping it up into discrete parts. The average value of [voltage](https://en.wikipedia.org/wiki/Volt" \o "Volt) (and [current](https://en.wikipedia.org/wiki/Electric_current" \o "Electric current)) fed to the [load](https://en.wikipedia.org/wiki/Electrical_load" \o "Electrical load) is controlled by turning the switch between supply and load on and off at a fast rate. The longer the switch is on compared to the off periods, the higher the total power supplied to the load. Along with [maximum power point tracking](https://en.wikipedia.org/wiki/Maximum_power_point_tracking" \o "Maximum power point tracking) (MPPT), it is one of the primary methods of reducing the output of solar panels to that which can be utilized by a battery. PWM is particularly suited for running inertial loads such as motors, which are not as easily affected by this discrete switching, because their inertia causes them to react slowly. The PWM switching frequency has to be high enough not to affect the load, which is to say that the resultant waveform perceived by the load must be as smooth as possible.

The rate (or frequency) at which the power supply must switch can vary greatly depending on load and application. For example, switching has to be done several times a minute in an electric stove; 100 or 120 [Hz](https://en.wikipedia.org/wiki/Hertz" \o "Hertz) (double of the [utility frequency](https://en.wikipedia.org/wiki/Utility_frequency" \o "Utility frequency)) in a [lamp dimmer](https://en.wikipedia.org/wiki/Dimmer" \o "Dimmer); between a few kilohertz (kHz) and tens of kHz for a motor drive; and well into the tens or hundreds of kHz in audio amplifiers and computer power supplies. The main advantage of PWM is that power loss in the switching devices is very low. When a switch is off there is practically no current, and when it is on and power is being transferred to the load, there is almost no voltage drop across the switch. Power loss, being the product of voltage and current, is thus in both cases close to zero. PWM also works well with digital controls, which, because of their on/off nature, can easily set the needed duty cycle. PWM has also been used in certain [communication systems](https://en.wikipedia.org/wiki/Signalling_(telecommunication)" \o "Signalling (telecommunication)) where its duty cycle has been used to convey information over a communications channel.

In electronics, many modern [microcontrollers](https://en.wikipedia.org/wiki/Microcontroller" \o "Microcontroller) (MCUs) integrate PWM controllers exposed to external pins as peripheral devices under [firmware](https://en.wikipedia.org/wiki/Firmware" \o "Firmware) control by means of internal programming interfaces. These are commonly used for [direct current](https://en.wikipedia.org/wiki/Direct_current" \o "Direct current) (DC) [motor control](https://en.wikipedia.org/wiki/Motor_controller" \o "Motor controller) in [robotics](https://en.wikipedia.org/wiki/Robotics" \o "Robotics) and other applications.



Figure 5: An example of PWM in an idealized inductor driven by a (blue) voltage source modulated as a series of pulses, resulting in a (red) sine-like current in the inductor. The rectangular voltage pulses nonetheless result in a more and more smooth current waveform, as the switching frequency increases. Note that the current waveform is the integral of the voltage waveform.

The resulting voltage thus depends on the duty cycle:

Where:

D - duty cycle

PW - pulse width

T - Total period

#### Theory: Differential Steering

The AGV move not only in a straight line but they also have to execute some rotation with different angles. Of course, robot navigation accuracy is greatly degraded when rotation is imprecise. The robot rotation angles were determined by a simple mathematical method. If the radius of robot wheel, the length of the horizontal axis (distance of two wheels) and the angle of wheel rotation are known, the rotation angle of robot (virtually, the axial rotation angle which connects the two wheels) can be simply determined as in the following figure.

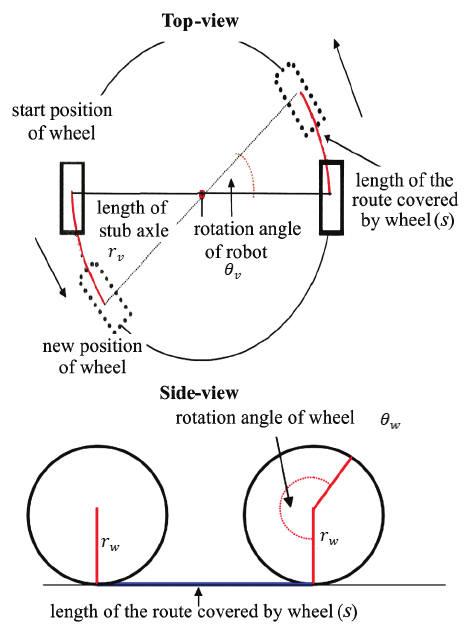


Figure 6: How the wheel rotation (opposite directions) affects vehicle rotation

Where:

s − length of the route covered by the wheel

rw − radius of wheel

rv − length of vehicle stub axle

w − rotation angle of wheel and

v − rotation angle of vehicle (axial)

Next if we take into angular velocity of the wheel and rotation time into account, and let the rotation angle of the robot be the desired rotation angle of the user,

Where:

t - rotation time

w - motor/wheel angular velocity

Within the following code this formula is implemented, albeit with different symbols for the variables. These are explained in the comments in the code.

#### Coding

The codes can be found in the appendix and also <https://github.com/ZHANG-Fengchu/AGV-MOTOR-CONTROL>, where the codes are the following:

[MDD10A.py](https://github.com/ZHANG-Fengchu/AGV-MOTOR-CONTROL/blob/master/MDD10A.py" \o "MDD10A.py)

[AGV\_MOVEMENT.py](https://github.com/ZHANG-Fengchu/AGV-MOTOR-CONTROL/blob/master/AGV_MOVEMENT.py" \o "AGV_MOVEMENT.py)

[TEST\_MDD10A.py](https://github.com/ZHANG-Fengchu/AGV-MOTOR-CONTROL/blob/master/TEST_MDD10A.py" \o "TEST_MDD10A.py)

Where [AGV\_MOVEMENT.py](https://github.com/ZHANG-Fengchu/AGV-MOTOR-CONTROL/blob/master/AGV_MOVEMENT.py" \o "AGV_MOVEMENT.py) is the main code for our purposes; MDD10A.py can be seen as the “library” or “software driver” for the MDD10A motor driver; and TEST\_MDD10A.py is the original code written by [Mohammed Omar Salameh](https://www.youtube.com/channel/UCubYb4lIdrh09Vme9GFtvCA) and is only there for reference and can be ignored. In addition to the comments already in the code itself, the first two codes will be explained briefly here in this report.

**[MDD10A.py](https://github.com/ZHANG-Fengchu/AGV-MOTOR-CONTROL/blob/master/MDD10A.py" \o "MDD10A.py)**

Essentially this code manages how information from the GPIO pins from the Raspberry Pi is interpreted into PWM electrical signal that further controls the movement of the motors. For the end user, most of the code can be left as it, with only the following lines of interest that can be modified or optimized if required:

|  |  |
| --- | --- |
| 26 | leftMotor\_DIR\_pin = 22 |
| 28 | rightMotor\_DIR\_pin = 23 |
| 33 | leftMotor\_PWM\_pin = 17 |
| 34 | rightMotor\_PWM\_pin = 18 |

The above four lines of code reflect the connection to the Pi, as seen previously in the circuit configuration. The numbers are simply the GPIO pin number that are connected to, and can be modified here if the user wishes to use other GPIO pins instead.

|  |  |
| --- | --- |
| 38 | leftMotorPWM = io.PWM(leftMotor\_PWM\_pin,1000) |
| 39 | rightMotorPWM = io.PWM(rightMotor\_PWM\_pin,1000) |

The above 2 lines of code modifies the PWM frequency of the two respective motors. This frequency affects how “smooth” the motors turn (and does not affect angular velocity), where the higher the frequency the less jerky the turning. According to the data sheet this frequency can be set to maximum 20000Hz. In practice, any value over 500Hz would be smooth enough that the jerkiness can not be visible to the naked eye.

**[AGV\_MOVEMENT.py](https://github.com/ZHANG-Fengchu/AGV-MOTOR-CONTROL/blob/master/AGV_MOVEMENT.py" \o "AGV_MOVEMENT.py)**

Due to the lack of python experience, this code is quite verbose and lacks efficient use of functions. However it works, and each block will be explained briefly in addition to the existing comments already existing within the code.

|  |  |
| --- | --- |
| 7 | import MDD10A as HBridge |
| 10 | import RPi.GPIO as GPIO |

These two lines import the two most important libraries: “MDD10A” as the driver, and RPi.GPIO to control the pins of the Pi.

|  |  |
| --- | --- |
| 15  30  35 | # Set up sensors  # Initialize values  # Default vehicle parameters if no user input. |

Under these comments lines are code that set up the sensor parameters so that the information input and output into them can be processed.

Under Initialize values and default parameters are code that set up data and equations for further calculation based on the mathematical theory of differential steering.

|  |  |
| --- | --- |
| 50 | # The Menu |

The menu gives a list of actions the user can perform by keying in particular characters.

|  |  |
| --- | --- |
| 83 | # Main program |

Here is where the associated actions of the input character are processed.

|  |  |
| --- | --- |
| 90  112 | if(char=="p"):  speedleft = 0.0  speedright = 0.0  HBridge.setMotorLeft(speedleft)  HBridge.setMotorRight(speedright)  print("Please enter half of distance between the two wheels in meters.")  rv=float(input())  print("Please enter radius of robot wheel in meters")  rw=float(input())  print("Please enter wheel motor PWM duty cycle, from 0.1 to 1.0 (i.e. 10% to 100%).")  print("Note that the absolute value of the average of the measured angular velocity of each motor at 100% duty cycle is",max\_omega\_w,"rads^-1.")  duty\_cycle=float(input())  omega\_w=max\_omega\_w\*duty\_cycle  print("Please enter desired vehicle rotation in degrees, for double wheel turning.")  theta\_v\_deg=float(input())  theta\_v\_rad=theta\_v\_deg\*(math.pi)/180  rotatetime=int((theta\_v\_rad/omega\_w)\*(rv/rw))  print("Please enter desired movement distance in meters, for straight forward and backward movement only.")  dis=float(input())  movetime=int(dis/(omega\_w\*rw))  print("Decide Ultrasound sensor distance threshold.")  us\_threshold=float(input())  printscreen() |

From line 90 to 112, the code can modify the mathematical formula parameters by user input.

|  |  |
| --- | --- |
| 115  143 | if(char == "f"):  speedleft = duty\_cycle  speedright = duty\_cycle  print("========== Current motor status ==========")  print("current left motor duty cycle:",speedleft)  print("current right motor duty cycle:",speedright)  print("Will take:",movetime,"seconds")  print("Press ctrl-c to terminate movement.")  time\_at\_start = time.time()  time\_paused = 0  time\_elapsed = int((time.time() - time\_at\_start))   try:   while(time\_elapsed < movetime):  ir\_sensor=GPIO.input(25)  time\_elapsed = int((time.time() - time\_at\_start)) - time\_paused  if ir\_sensor==1 and us\_sensor.value>us\_threshold:  HBridge.setMotorLeft(speedleft)  HBridge.setMotorRight(speedright)  print("Time elapsed:",time\_elapsed,"seconds; No obstacles; US sensor value:",us\_sensor.value,"meters"+ "\r", end="")  elif ir\_sensor==0 or us\_sensor.value<us\_threshold:  HBridge.setMotorLeft(0)  HBridge.setMotorRight(0)  time.sleep(1)  time\_paused = time\_paused + 1  print("Time elapsed:",time\_elapsed,"seconds; Yes obstacle; US sensor value:",us\_sensor.value,"meters; Time paused:", time\_paused, "seconds" + "\r", end="")  except KeyboardInterrupt:  pass  char = "q"  print("Time's up") |

From line 115 to 143, this defines the forward movement. (Note that the logic for backwards movement, clockwise and anticlockwise rotation is identical.)

1. First the duty cycle, and thus angular velocity of the motors are set to the value of the current duty cycle variable,
2. Then the current time is stored in the variable time\_at\_start
3. Then a new variable time\_paused is declared for storage of time paused when the sensor detects an object.
4. Then a variable time\_elapsed is declared to store the time the AGV has already moved disregarding time paused.
5. These three variable are made into a “timer”-like usage when used with the movetime variable.
6. Within try: a while loop is declared, and will run unless KeyboardInterrupt: happens, which is the keypress CTRL-C, which will terminate the action immediately before the movement is finished.
7. The while loop runs as long as the time\_elapsed does not exceed movetime (the time the AGV is supposed to move.
8. If the sensors detect an object, time\_paused starts counting.
9. This time\_paused value offsets the time\_elapsed during the time there is an obstacle detected.
10. When the obstacle is removed, the time\_elapsed is no longer offset by time\_paused and continues to count up until movetime.
11. The character “q” is set for the variable char, which stops the motors.

The rest of the code is straightforward and requires no explanation.

#### Testing

Because of the lack of available stress testing environment or tools, testing was not available to be conducted for the feasibility of the design. However, the motors behaved as expected, which can be seen in the video in the next section.

The only test that was conducted was measuring the angular velocity of the motors at full battery voltage 22.2V and max duty cycle, which yielded an average of 18.43 rad/s. Since the angular velocity drops with battery voltage as it drains, and the motors were not encoded, this created a problem that the duty cycle cannot self-adjust to compensate when. This will be discussed in the “room for improvement” section.

#### Prototyping

Due to the lack of access to resources caused by the lockdown, a simple AGV-shaped “vehicle” was made out of cardboard and suitcase wheels, just to get the idea of how it may finally look. The following video shows a demo of the “prototype”:

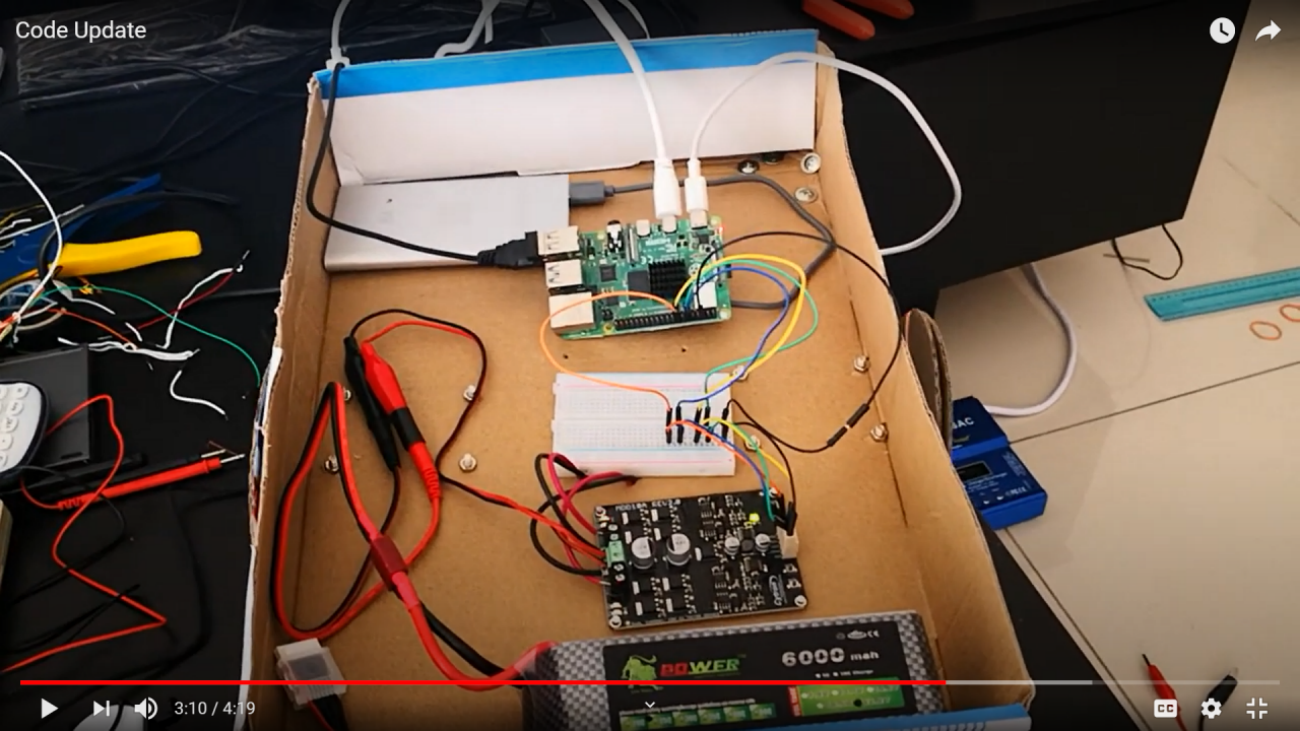


Figure 7: Screenshot from <https://www.youtube.com/watch?v=k0SI_MBrwKk>

The next video shows testing of the sensors:

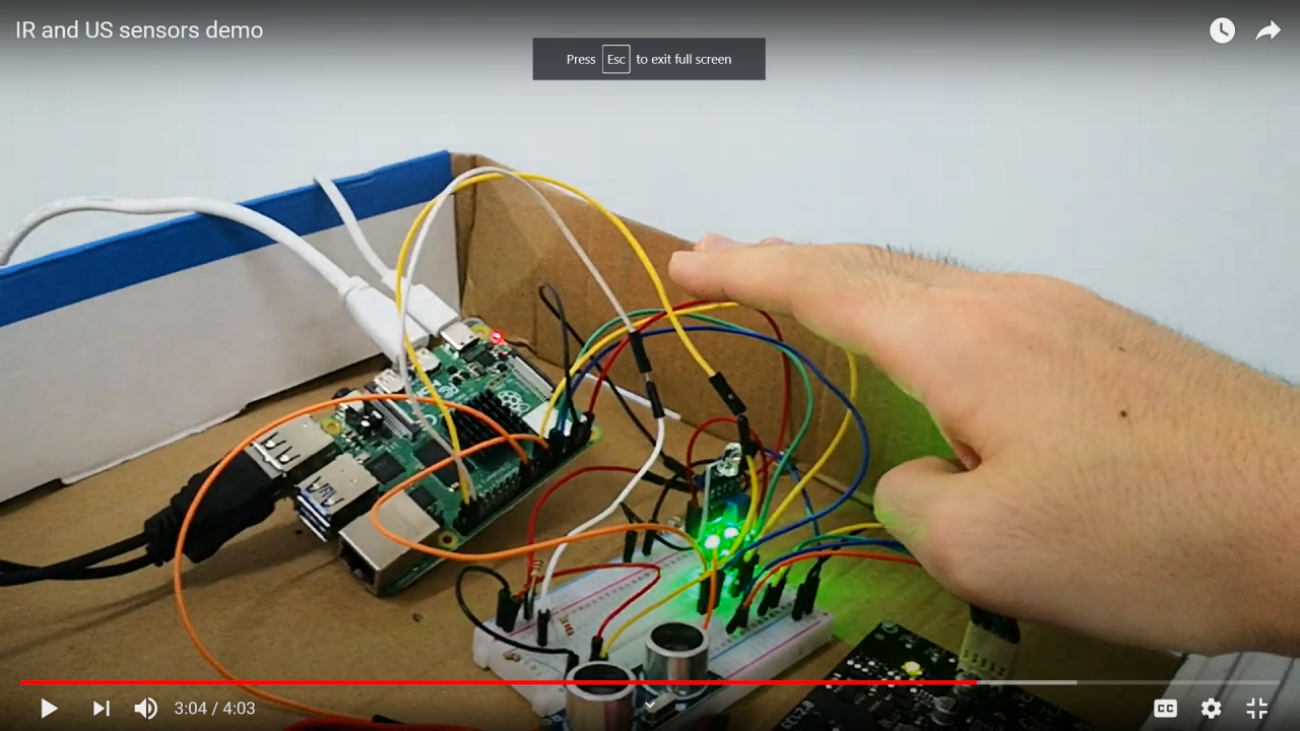


Figure 8: Screenshot from <https://www.youtube.com/watch?v=k0SI_MBrwKk>

### 3.2.c Critical Analysis

**Assumptions**

* All the purchased electronic components are compatible with each other.
* Sufficient online resources for research and study.

**Methodological Limitations**

* Lack of access to laboratory, and thus tools needed for testing and prototyping.
* Lack of sufficient knowledge of python to fully design all the code.

**Limitations to Applicability**

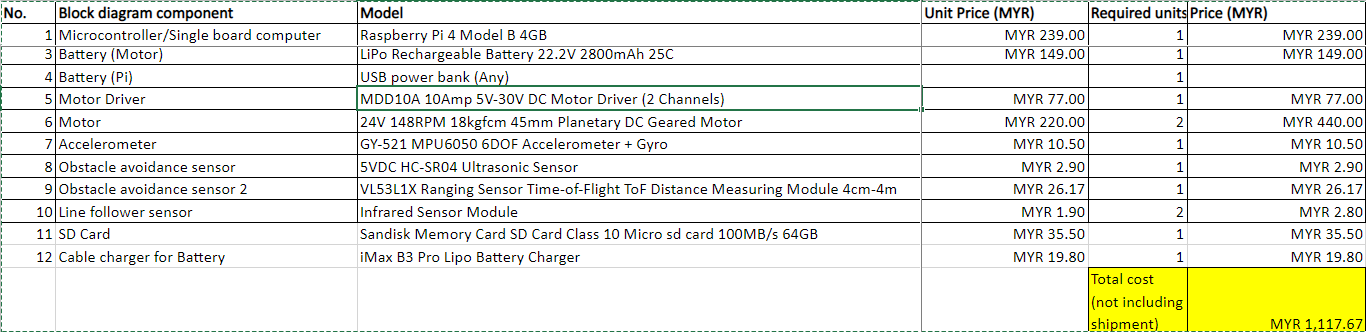
Due to limitations the applicability is unknown and untested.

**Replicability of Results**

By reading this report and the associated references the reader can replicate the current stage of testing and prototyping.

## 3.3 Costs

Table 1: List of components purchased and their cost.



The spreadsheet is also available at this link: <https://1drv.ms/x/s!AnnV65ZwDIZEnwu_Bgon4ZY_pzgN?e=SFLH7a>

## 3.4 Project Management and Workflow

The project was split into smaller tasks which were then delegated to the two interns to work on separately, with the original plan to integrate the work together during prototyping. Unfortunately due to the limitation this was not materialized.

Other tasks mostly involving algorithm programming was proven too challenging and was delegated to the supervisor instead.

The original planned workflow:

Table 2: Gantt chart for overall workflow.



The spreadsheet is also available at this link:

<https://1drv.ms/x/s!AnnV65ZwDIZEnmsM5SFscyW57QRq?e=jpxDgr>

# Chapter 4: Problems Occurred and Description

## 4.1 What could have been be the next steps?

* Integration of all work done so far within one circuit and program.
* Design and/or purchasing of hardware such as wheels and casing to create a testable prototype.
* Replacement of components, such as the non-encoded motors with encoded ones, for information feedback and improvement of design.

## 4.2 What would we have done if resources weren’t limited?

* Outsource coding to professional freelancers.
* Using tools for prototyping casing such as 3D printers or laser wood cutters.
* More thorough discussion, interaction and learning from the industrial supervisor and the rest of the team at the workplace.

# Conclusion

In conclusion, although the original plan of building a functional and testable prototype did not come to fruition, progress had been made in the bulk of the design for QR detection, server communication, and motor control.

The current progress ensures what has been made can be easily integrated. The process of creating a functional prototype is entirely feasible and would hopefully be made easier after limitations of travel restrictions are lifted.

As for the internship as a whole, tremendous amount of real industry experience has been acquired, and a new perspective on what needs to be learned in order to be skillful in the workplace will act as a guide for the rest of the academic semesters.

# References

Cheng Hua

<https://www.linkedin.com/company/cheng-hua-engineering-works-sdn-bhd/>

https://www.chenghua-m.com/

D\* Lite:

[https://en.wikipedia.org/wiki/D\*#D\*\_Lite](https://en.wikipedia.org/wiki/D*#D*_Lite)

<https://atsushisakai.github.io/PythonRobotics/>

Motor control:

<https://en.wikipedia.org/wiki/Duty_cycle>

<https://en.wikipedia.org/wiki/Pulse-width_modulation>

<https://www.robotshop.com/community/forum/t/review-for-motor-driver-mdd10a-controlling-by-raspberry-pi/31740>

Differential steering:

<https://www.researchgate.net/publication/268076921_Gathering_simulation_of_real_robot_swarm>

Sensors:

<https://www.youtube.com/watch?v=hmgLxCKZBZg>

<https://www.youtube.com/watch?v=JvQKZXCYMUM>

# Appendix

## Appendix A: Meeting Minute Summaries

20210601

Introduction by industrial supervisor Mr Tan to the

company.

Brief explanation of the 3 phases of tasks we the interns

are expected to perform in these three months time. Phase

1 is basic knowledge of the warehouse process and

management architecture; Phase 2 is researching methods

of automating and improving the process of picking slow

movement items from the warehouse; Phase 3 is product

development.

20210602

A more in depth explanation into Phase 1 task: What is

the basic warehouse process? What is the warehouse

architecture? What is the relation between Warehouse

Management System, Warehouse Control System? What

are the modern electronic devices and equipment used in

warehouses?

20210603

No Zoom meeting. Preliminary reading on phase 1

research topic.

20210604

Explanation of simple findings about Phase 1 so far.

Invited to listen to a meeting where a company that

proposes Bluetooth tracking systems to Cheng Hua to

replace GPS tracking.

20210605 --

20210608

No Zoom meetings. Further reading and research on

Phase 1 topic. Preparation for presentation and report.

20210609

Presentation for Phase 1.

Briefing for what to research for Phase 2, mainly success

stories for automated picking systems where the operator

or worker does not need to moving into the warehouse to

pick items.

20210610

No Zoom meeting. Preliminary research on Phase 2.

20210611

Short presentation for information found for Phase 2 so

far. Including the differentiation between Category A, B,

C items (fast, middle, slow moving items) and how they

are stocked in the warehouse, and case studies of

automated picking systems such as SQUID designed or

used by companies like Alibaba.

Decision by product development manager for us to

further research into the SQUID and the Quicktron

warehouse picking robots.

2021-06-16

Continued research into Phase 2. Presentation on the

introduction to mobile robots as used in warehouses:

AMRs and AGVs. Given case study examples such as Squid

by BionicHive.

2021-06-18

Feasibility studies on QR code navigation for AGVs in the

warehouse. Discussed cost task flow, and hardware

architecture for sensors, QR code, and the AGV itself.

Gave three examples of existing products/solutions that

use QR navigation.

2021-06-23

Beginning of Phase 3 - product design of QR navigation AGV.

General working principles of AGV and required electronic

hardware for prototyping.

2021-06-25

Price listing of hardware for prototyping, and suggested

algorithms, simulation software, and top level block

diagram for testing.

2021-06-30

Research into detection of QR codes from camera feed

using python code.

2021-07-02

Research into path planning algorithms. Explanation of

Dijkstra’s algorithm and A\* algorithm, and exploring the

feasibility of using them for our purpose.

2021-07-07

Explanation of the limitations of A\* algorithm. Exploring

incremental heuristic search algorithms, and suggestion of

using D\* Lite algorithm instead. Video demonstration of

the decision mathematical theory behind D\* Lite algorithm.

2021-07-09

Continued research into D\* Lite algorithm, and searching

for existing python code that could be modified and

integrated into our overall design.

2021-07-14

Discovery of complete set of python navigation codes on

GitHub, and reading through them to understand how they

work.

2021-07-16

Learned how to use GitKraken, and GitHub for version

management, and forking other users’ code.

Failed attempts to modify selected python codes for D\*

algorithm to suit our purposes, due to the required high

level coding skills

2021-07-21

Transition to research on motor control instead.

Preliminary research into using Raspberry Pi and the

MDD10A as PWM motor controllers.

2021-07-23

Research into differential steering, and learning how to

implement into python. Learning the physics and

mathematical formulas, and the relation between turning

time, desired angle, angular velocity, etc.

2021-07-30

Received AGV battery, motors, and MDD10A motor driver

from company.

Successfully connected the three components to verify

functionality.

Failed to use python code to control the motor driver

though due to coding errors.

2021-08-04

Successfully implemented angle rotation formula using

python code to achieve desired rotation time output.

2021-08-06

Created AGV-shaped prototype with cardboard integrating

all the components (Powerbank, Pi, motor driver, motors,

AGV battery, breadboard, jumper wires, and smaller

electronic components), that could function wirelessly and

headlessly.

Updated code to include commands for forward and

backwards movement.

Realized I should have recommended motors with

encoders, since the currents ones give no feedback on

current angular velocity, thus reducing rotation accuracy

due to dropping voltage from using the AGV battery.

## Appendix B: Codes

Can also be found on <https://github.com/ZHANG-Fengchu/AGV-MOTOR-CONTROL>

### MDD10A.py

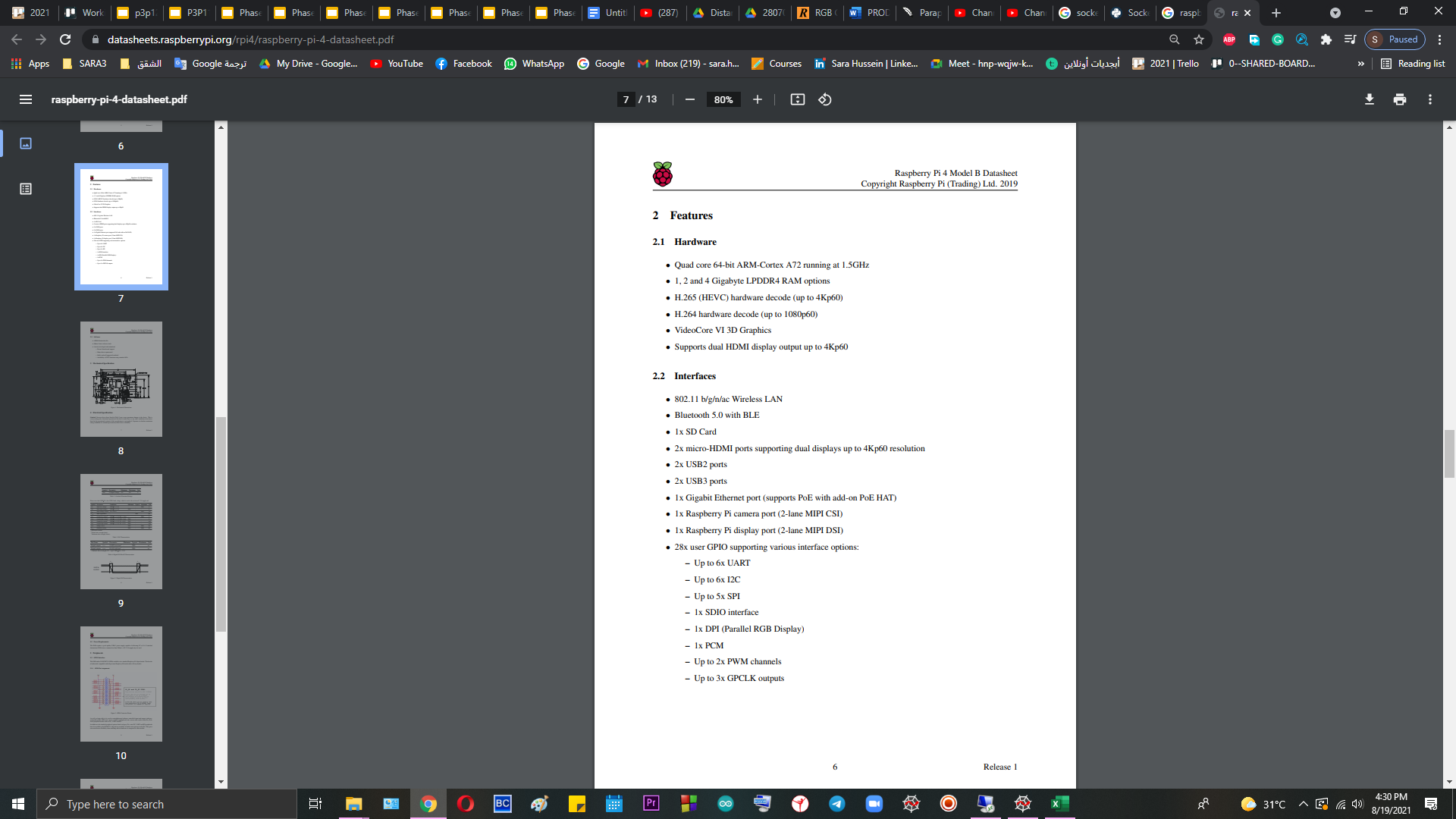
|  |
| --- |
| #!/usr/bin/env python |
|  | # coding: latin-1 |
|  | # I am Mohammad Omar, this module is builded to interface with the Driver MDD10A, to control two DC motors. |
|  | # the original code designed by Ingmar Stapel ,www.raspberry-pi-car.com to control two motors with a L298N H-Bridge |
|  | # The pins configuration for Model B Revision 1.0 |
|  | # How to Use this module: 1- creating an instance of the class. 2- call the Init function, 3- call commands functions |
|  | # Example: |
|  | # import MDD10A |
|  | # Motors = MDD10A.MDD10A() |
|  | # Import the libraries the class needs |
|  | import RPi.GPIO as io |
|  | io.setmode(io.BCM) |
|  | # Constant values, |
|  | PWM\_MAX = 100 |
|  | # Disable warning from GPIO |
|  | io.setwarnings(False) |
|  | # Here we configure the GPIO settings for the left and right motors spinning direction. |
|  | # as described in the user manual of MDD10A https://www.robotshop.com/media/files/pdf/user-manual-mdd10a.pdf |
|  | # there are four input PWM1-DIR1-PWM2-DIR2 |
|  | # WITH MAX Frequency 20 Hz, and it works as follow, |
|  | # Input DIR Output-A Output-B |
|  | # PWM off X off off |
|  | # PWM on off on off |
|  | # PWM on on off on |
|  | # The pins configuration for Model B Revision 1.0 |
|  | leftMotor\_DIR\_pin = 22 |
|  | io.setup(leftMotor\_DIR\_pin, io.OUT) |
|  | rightMotor\_DIR\_pin = 23 |
|  | io.setup(rightMotor\_DIR\_pin, io.OUT) |
|  | io.output(leftMotor\_DIR\_pin, False) |
|  | io.output(rightMotor\_DIR\_pin, False) |
|  | # Here we configure the GPIO settings for the left and right motors spinning speed. |
|  | leftMotor\_PWM\_pin = 17 |
|  | rightMotor\_PWM\_pin = 18 |
|  | io.setup(leftMotor\_PWM\_pin, io.OUT) |
|  | io.setup(rightMotor\_PWM\_pin, io.OUT) |
|  | # MAX Frequency 20 Hz |
|  | leftMotorPWM = io.PWM(leftMotor\_PWM\_pin,1000) |
|  | rightMotorPWM = io.PWM(rightMotor\_PWM\_pin,1000) |
|  | leftMotorPWM.start(0) |
|  | leftMotorPWM.ChangeDutyCycle(0) |
|  | rightMotorPWM.start(0) |
|  | rightMotorPWM.ChangeDutyCycle(0) |
|  | leftMotorPower = 0 |
|  | rightMotorPower = 0 |
|  | def getMotorPowers(): |
|  | return (leftMotorPower,rightMotorPower) |
|  | def setMotorLeft(power): |
|  | # SetMotorLeft(power) |
|  | # Sets the drive level for the left motor, from +1 (max) to -1 (min). |
|  | # This is a short explanation for a better understanding: |
|  | # SetMotorLeft(0) -> left motor is stopped |
|  | # SetMotorLeft(0.75) -> left motor moving forward at 75% power |
|  | # SetMotorLeft(-0.5) -> left motor moving reverse at 50% power |
|  | # SetMotorLeft(1) -> left motor moving forward at 100% power |
|  | if power < 0: |
|  | # Reverse mode for the left motor |
|  | io.output(leftMotor\_DIR\_pin, False) |
|  | pwm = -int(PWM\_MAX \* power) |
|  | if pwm > PWM\_MAX: |
|  | pwm = PWM\_MAX |
|  | elif power > 0: |
|  | # Forward mode for the left motor |
|  | io.output(leftMotor\_DIR\_pin, True) |
|  | pwm = int(PWM\_MAX \* power) |
|  | if pwm > PWM\_MAX: |
|  | pwm = PWM\_MAX |
|  | else: |
|  | # Stopp mode for the left motor |
|  | io.output(leftMotor\_DIR\_pin, False) |
|  | pwm = 0 |
|  | # print "SetMotorLeft", pwm |
|  | leftMotorPower = pwm |
|  | leftMotorPWM.ChangeDutyCycle(pwm) |
|  | def setMotorRight(power): |
|  | # SetMotorRight(power) |
|  | # Sets the drive level for the right motor, from +1 (max) to -1 (min). |
|  | # This is a short explanation for a better understanding: |
|  | # SetMotorRight(0) -> right motor is stopped |
|  | # SetMotorRight(0.75) -> right motor moving forward at 75% power |
|  | # SetMotorRight(-0.5) -> right motor moving reverse at 50% power |
|  | # SetMotorRight(1) -> right motor moving forward at 100% power |
|  | if power < 0: |
|  | # Reverse mode for the right motor |
|  | io.output(rightMotor\_DIR\_pin, True) |
|  | pwm = -int(PWM\_MAX \* power) |
|  | if pwm > PWM\_MAX: |
|  | pwm = PWM\_MAX |
|  | elif power > 0: |
|  | # Forward mode for the right motor |
|  | io.output(rightMotor\_DIR\_pin, False) |
|  | pwm = int(PWM\_MAX \* power) |
|  | if pwm > PWM\_MAX: |
|  | pwm = PWM\_MAX |
|  | else: |
|  | # Stopp mode for the right motor |
|  | io.output(rightMotor\_DIR\_pin, False) |
|  | pwm = 0 |
|  | # print "SetMotorRight", pwm |
|  | rightMotorPower = pwm |
|  | rightMotorPWM.ChangeDutyCycle(pwm) |
|  | def exit(): |
|  | # Program will clean up all GPIO settings and terminates |
|  | io.output(leftMotor\_DIR\_pin, False) |
|  | io.output(rightMotor\_DIR\_pin, False) |
|  | io.cleanup() |

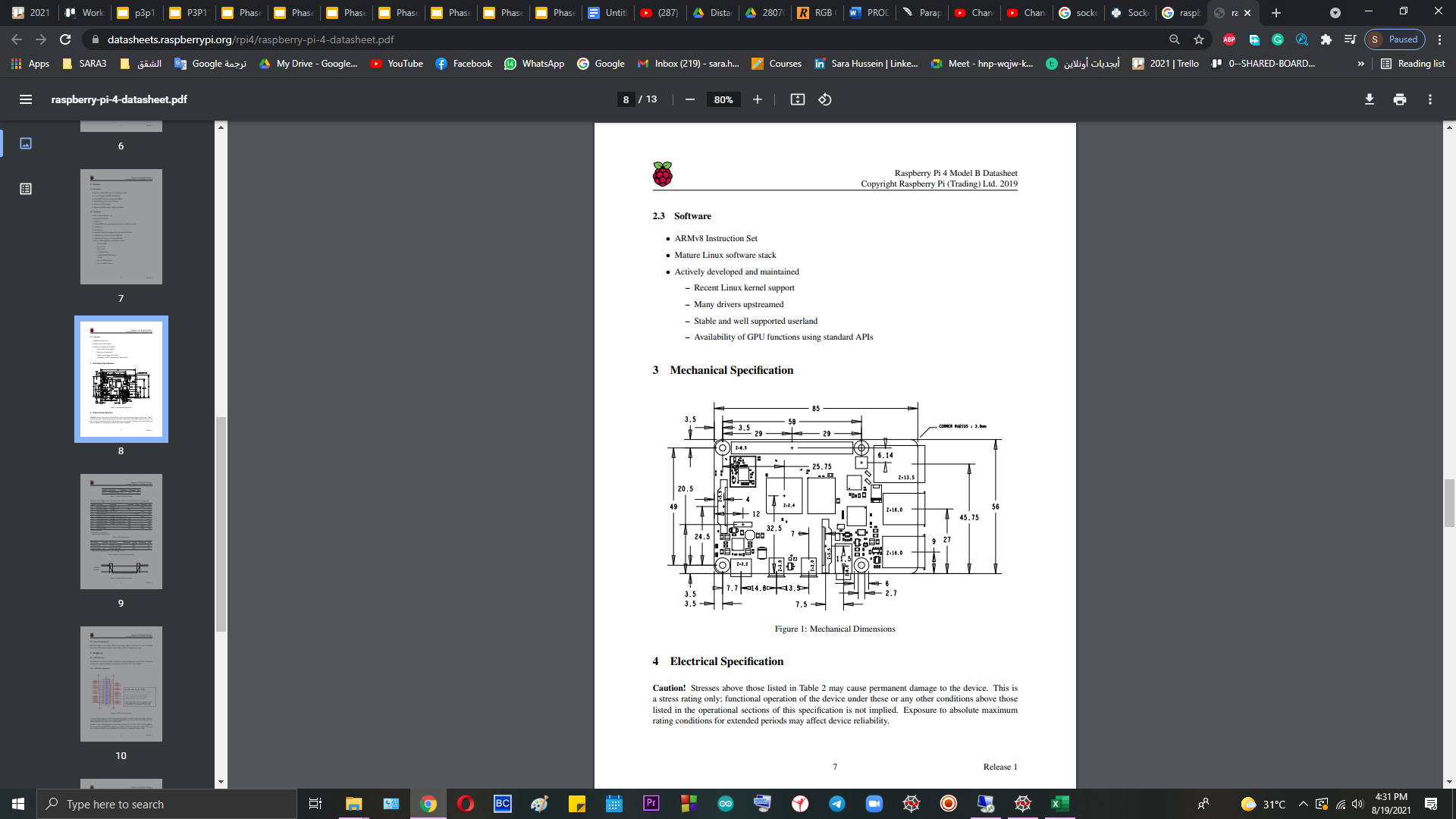
### AGV\_MOVEMENT.py

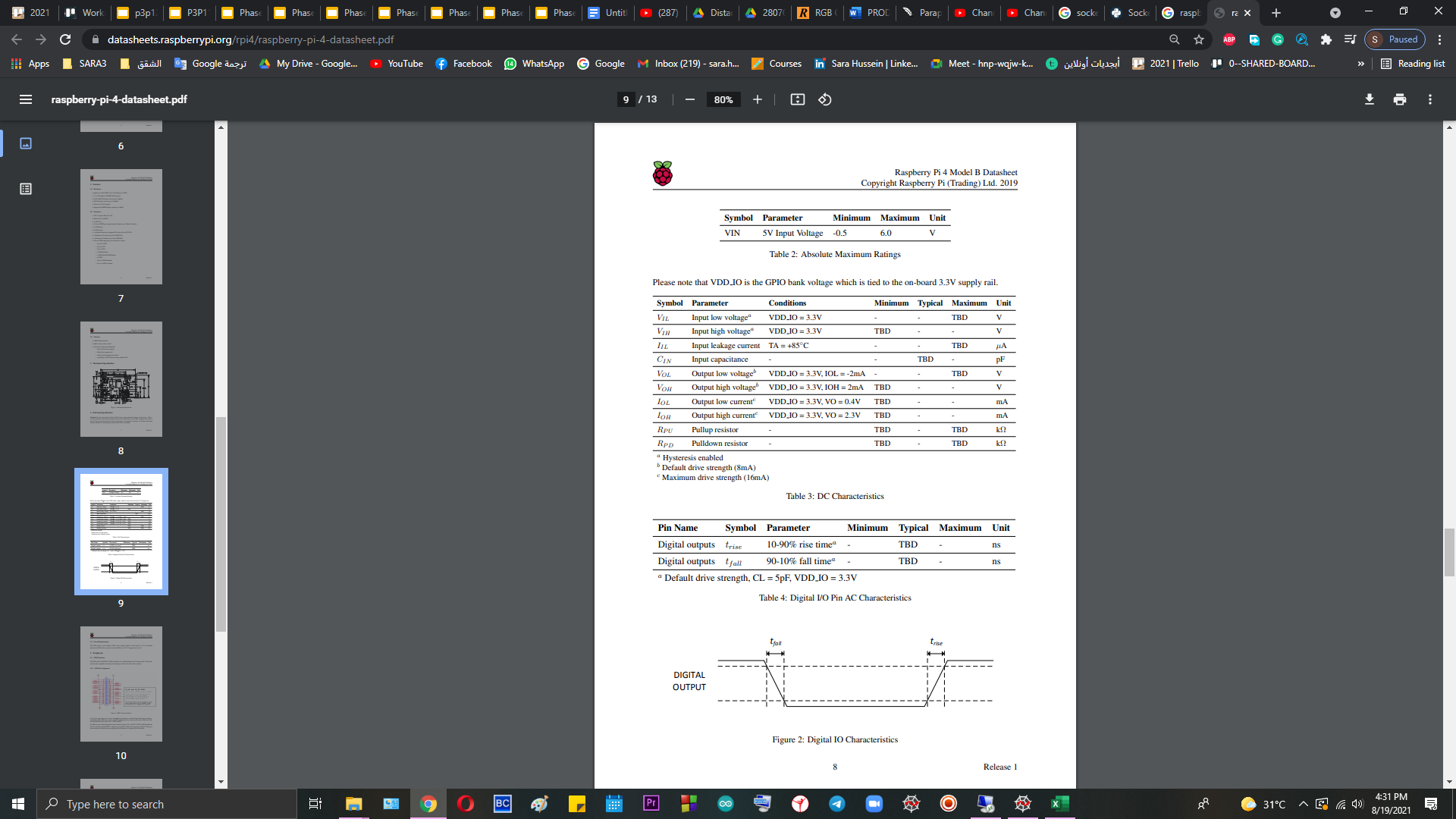
|  |
| --- |
| # Code written by ZHANG Fengchu, inspired by code written by Mohammed Omar Salameh. |
|  | # Code to be used with MDD10A.py. |
|  | # Purpose of code is to have the motors rotate at a given speed for a certain amount of time, dependent on user given parameter values. |
|  | # Whether the motors rotates in the same direction or in opposite directions depends |
|  | # on whether the command from the user is for the vehicle to move forwards or rotate. |
|  | import sys, tty, termios, os |
|  | import MDD10A as HBridge |
|  | import time |
|  | from time import sleep |
|  | import RPi.GPIO as GPIO |
|  | import math |
|  | from signal import signal, SIGTERM, SIGHUP, pause |
|  | from gpiozero import DistanceSensor |
|  |  |
|  | # Set up sensors |
|  | ## IR sensor |
|  | GPIO.setmode(GPIO.BCM) |
|  | GPIO.setup(25, GPIO.IN) |
|  |  |
|  | ## US sensor |
|  | us\_sensor = DistanceSensor(echo=20, trigger=21) |
|  | def safe\_exit(signum, frame): |
|  | exit(1) |
|  |  |
|  | us\_threshold = 0.2 |
|  |  |
|  | signal(SIGTERM, safe\_exit) |
|  | signal(SIGHUP, safe\_exit) |
|  |  |
|  | # Initialize values |
|  | max\_omega\_w=18.43 |
|  | speedleft=0.0 |
|  | speedright=0.0 |
|  |  |
|  | # Default vehicle parameters if no user input. |
|  | rv=0.5 |
|  | rw=0.1 |
|  | duty\_cycle=0.1 |
|  | omega\_w=max\_omega\_w\*duty\_cycle |
|  | theta\_v\_deg=360 |
|  | theta\_v\_rad=theta\_v\_deg\*(math.pi)/180 |
|  | rotatetime=int((theta\_v\_rad/omega\_w)\*(rv/rw)) |
|  | dis=2 |
|  | movetime=int(dis/(omega\_w\*rw)) |
|  |  |
|  | # User raw input (for now) to determine how the user wishes for the robot to move. |
|  | def getch(): |
|  | return input() |
|  |  |
|  | # The Menu |
|  | def printscreen(): |
|  | os.system('clear') |
|  | print("+: Right motor speed up") |
|  | print("-: Right motor speed down") |
|  | print(".: Left motor speed up") |
|  | print(",: Left motor speed down") |
|  | print("p: set vehicle parameters") |
|  | print("f: move forwards (default 1 meter)") |
|  | print("b: move backwards (defautl 1 meter)") |
|  | print("r: rotate clockwise (default 90 degrees)") |
|  | print("l: rotate anticlockwise (default 90 degrees)") |
|  | print("q: stops the motors") |
|  | print("x: exit") |
|  | print("========== Current motor status ==========") |
|  | print("current left motor duty cycle:",speedleft) |
|  | print("current right motor duty cycle:",speedright) |
|  | print("========== Current user parameters ==========") |
|  | print("half of track length:",rv,"meters") |
|  | print("radius of wheel:",rw,"meters") |
|  | print("user duty cycle:",duty\_cycle) |
|  | print("user input wheel angular velocity:",omega\_w,"radians per second") |
|  | print("desired rotation angle in degrees:",theta\_v\_deg,"degrees") |
|  | print("desired rotation angle in radians:",theta\_v\_rad,"radians") |
|  | print("estimated rotation time based on current parameters:",rotatetime,"seconds") |
|  | print("desired straight movement distance:",dis,"meters") |
|  | print("estimated straight movement time based on current parameters:",movetime,"seconds") |
|  | print("current ultrasound sensor distance threshold:",us\_threshold,"meters") |
|  |  |
|  | # Welcome text |
|  | print("Welcome, press one of the following characters.") |
|  | printscreen() |
|  |  |
|  | # Main program |
|  | while True: |
|  | # Keyboard character retrieval method. This method will save |
|  | # the pressed key into the variable char |
|  | char = getch() |
|  |  |
|  | # Require user to input parameters of robot. |
|  | if(char=="p"): |
|  | speedleft = 0.0 |
|  | speedright = 0.0 |
|  | HBridge.setMotorLeft(speedleft) |
|  | HBridge.setMotorRight(speedright) |
|  | print("Please enter half of distance between the two wheels in meters.") |
|  | rv=float(input()) |
|  | print("Please enter radius of robot wheel in meters") |
|  | rw=float(input()) |
|  | print("Please enter wheel motor PWM duty cycle, from 0.1 to 1.0 (i.e. 10% to 100%).") |
|  | print("Note that the absolute value of the average of the measured angular velocity of each motor at 100% duty cycle is",max\_omega\_w,"rads^-1.") |
|  | duty\_cycle=float(input()) |
|  | omega\_w=max\_omega\_w\*duty\_cycle |
|  | print("Please enter desired vehicle rotation in degrees, for double wheel turning.") |
|  | theta\_v\_deg=float(input()) |
|  | theta\_v\_rad=theta\_v\_deg\*(math.pi)/180 |
|  | rotatetime=int((theta\_v\_rad/omega\_w)\*(rv/rw)) |
|  | print("Please enter desired movement distance in meters, for straight forward and backward movement only.") |
|  | dis=float(input()) |
|  | movetime=int(dis/(omega\_w\*rw)) |
|  | print("Decide Ultrasound sensor distance threshold.") |
|  | us\_threshold=float(input()) |
|  | printscreen() |
|  |  |
|  | # The vehicle will move forwards to the user desired distance when f is pressed. |
|  | if(char == "f"): |
|  | speedleft = duty\_cycle |
|  | speedright = duty\_cycle |
|  | print("========== Current motor status ==========") |
|  | print("current left motor duty cycle:",speedleft) |
|  | print("current right motor duty cycle:",speedright) |
|  | print("Will take:",movetime,"seconds") |
|  | print("Press ctrl-c to terminate movement.") |
|  | time\_at\_start = time.time() |
|  | time\_paused = 0 |
|  | time\_elapsed = int((time.time() - time\_at\_start)) |
|  | try: |
|  | while(time\_elapsed < movetime): |
|  | ir\_sensor=GPIO.input(25) |
|  | time\_elapsed = int((time.time() - time\_at\_start)) - time\_paused |
|  | if ir\_sensor==1 and us\_sensor.value>us\_threshold: |
|  | HBridge.setMotorLeft(speedleft) |
|  | HBridge.setMotorRight(speedright) |
|  | print("Time elapsed:",time\_elapsed,"seconds; No obstacles; US sensor value:",us\_sensor.value,"meters"+ "\r", end="") |
|  | elif ir\_sensor==0 or us\_sensor.value<us\_threshold: |
|  | HBridge.setMotorLeft(0) |
|  | HBridge.setMotorRight(0) |
|  | time.sleep(1) |
|  | time\_paused = time\_paused + 1 |
|  | print("Time elapsed:",time\_elapsed,"seconds; Yes obstacle; US sensor value:",us\_sensor.value,"meters; Time paused:", time\_paused, "seconds" + "\r", end="") |
|  | except KeyboardInterrupt: |
|  | pass |
|  | char = "q" |
|  | print("Time's up") |
|  |  |
|  | # The vehicle will move backwards to the user desired distance when b is pressed. |
|  | if(char == "b"): |
|  | speedleft = -duty\_cycle |
|  | speedright = -duty\_cycle |
|  | print("========== Current motor status ==========") |
|  | print("current left motor duty cycle:",speedleft) |
|  | print("current right motor duty cycle:",speedright) |
|  | print("Will take:",movetime,"seconds") |
|  | print("Press ctrl-c to terminate movement.") |
|  | time\_at\_start = time.time() |
|  | time\_paused = 0 |
|  | time\_elapsed = int((time.time() - time\_at\_start)) |
|  | try: |
|  | while(time\_elapsed < movetime): |
|  | ir\_sensor=GPIO.input(25) |
|  | time\_elapsed = int((time.time() - time\_at\_start)) - time\_paused |
|  | if ir\_sensor==1 and us\_sensor.value>us\_threshold: |
|  | HBridge.setMotorLeft(speedleft) |
|  | HBridge.setMotorRight(speedright) |
|  | print("Time elapsed:",time\_elapsed,"seconds; No obstacles; US sensor value:",us\_sensor.value,"meters"+ "\r", end="") |
|  | elif ir\_sensor==0 or us\_sensor.value<us\_threshold: |
|  | HBridge.setMotorLeft(0) |
|  | HBridge.setMotorRight(0) |
|  | time.sleep(1) |
|  | time\_paused = time\_paused + 1 |
|  | print("Time elapsed:",time\_elapsed,"seconds; Yes obstacle; US sensor value:",us\_sensor.value,"meters; Time paused:", time\_paused, "seconds" + "\r", end="") |
|  | except KeyboardInterrupt: |
|  | pass |
|  | char = "q" |
|  | print("Time's up") |
|  |  |
|  | # The vehicle will rotate clockwise to the user desired angle when r is pressed. |
|  | if(char == "r"): |
|  | speedleft = duty\_cycle |
|  | speedright = -duty\_cycle |
|  | print("========== Current motor status ==========") |
|  | print("current left motor duty cycle:",speedleft) |
|  | print("current right motor duty cycle:",speedright) |
|  | print("Will take:",rotatetime,"seconds") |
|  | print("Press ctrl-c to terminate movement.") |
|  | time\_at\_start = time.time() |
|  | time\_paused = 0 |
|  | time\_elapsed = int((time.time() - time\_at\_start)) |
|  | try: |
|  | while(time\_elapsed < rotatetime): |
|  | ir\_sensor=GPIO.input(25) |
|  | time\_elapsed = int((time.time() - time\_at\_start)) - time\_paused |
|  | if ir\_sensor==1 and us\_sensor.value>us\_threshold: |
|  | HBridge.setMotorLeft(speedleft) |
|  | HBridge.setMotorRight(speedright) |
|  | print("Time elapsed:",time\_elapsed,"seconds; No obstacles; US sensor value:",us\_sensor.value,"meters"+ "\r", end="") |
|  | elif ir\_sensor==0 or us\_sensor.value<us\_threshold: |
|  | HBridge.setMotorLeft(0) |
|  | HBridge.setMotorRight(0) |
|  | time.sleep(1) |
|  | time\_paused = time\_paused + 1 |
|  | print("Time elapsed:",time\_elapsed,"seconds; Yes obstacle; US sensor value:",us\_sensor.value,"meters; Time paused:", time\_paused, "seconds" + "\r", end="") |
|  | except KeyboardInterrupt: |
|  | pass |
|  | char = "q" |
|  | print("Time's up") |
|  |  |
|  | # The vehicle will rotate clockwise to the user desired angle when l is pressed. |
|  | if(char == "l"): |
|  | speedleft = -duty\_cycle |
|  | speedright = duty\_cycle |
|  | print("========== Current motor status ==========") |
|  | print("current left motor duty cycle:",speedleft) |
|  | print("current right motor duty cycle:",speedright) |
|  | print("Will take:",rotatetime,"seconds") |
|  | print("Press ctrl-c to terminate movement.") |
|  | time\_at\_start = time.time() |
|  | time\_paused = 0 |
|  | time\_elapsed = int((time.time() - time\_at\_start)) |
|  | try: |
|  | while(time\_elapsed < rotatetime): |
|  | ir\_sensor=GPIO.input(25) |
|  | time\_elapsed = int((time.time() - time\_at\_start)) - time\_paused |
|  | if ir\_sensor==1 and us\_sensor.value>us\_threshold: |
|  | HBridge.setMotorLeft(speedleft) |
|  | HBridge.setMotorRight(speedright) |
|  | print("Time elapsed:",time\_elapsed,"seconds; No obstacles; US sensor value:",us\_sensor.value,"meters"+ "\r", end="") |
|  | elif ir\_sensor==0 or us\_sensor.value<us\_threshold: |
|  | HBridge.setMotorLeft(0) |
|  | HBridge.setMotorRight(0) |
|  | time.sleep(1) |
|  | time\_paused = time\_paused + 1 |
|  | print("Time elapsed:",time\_elapsed,"seconds; Yes obstacle; US sensor value:",us\_sensor.value,"meters; Time paused:", time\_paused, "seconds" + "\r", end="") |
|  | except KeyboardInterrupt: |
|  | pass |
|  | char = "q" |
|  | print("Time's up") |
|  |  |
|  | # Speed up right motor |
|  | if(char == "+"): |
|  | speedright = speedright + 0.1 |
|  | if speedright > 1: |
|  | speedright = 1 |
|  | HBridge.setMotorRight(speedright) |
|  | printscreen() |
|  |  |
|  | # Slow down right motor |
|  | if(char == "-"): |
|  | speedright = speedright - 0.1 |
|  | if speedright < -1: |
|  | speedright = -1 |
|  | HBridge.setMotorRight(speedright) |
|  | printscreen() |
|  |  |
|  | # Speed up left motor |
|  | if(char == "."): |
|  | speedleft = speedleft + 0.1 |
|  | if speedleft > 1: |
|  | speedleft = 1 |
|  | HBridge.setMotorLeft(speedleft) |
|  | printscreen() |
|  |  |
|  | # Slow down left motor |
|  | if(char == ","): |
|  | speedleft = speedleft - 0.1 |
|  | if speedleft < -1: |
|  | speedleft = -1 |
|  | HBridge.setMotorLeft(speedleft) |
|  | printscreen() |
|  |  |
|  | # Stop the motors |
|  | if(char == "q"): |
|  | speedleft = 0 |
|  | speedright = 0 |
|  | HBridge.setMotorLeft(speedleft) |
|  | HBridge.setMotorRight(speedright) |
|  | printscreen() |
|  |  |
|  | # The "x" key will break the loop and exit the program |
|  | if(char == "x"): |
|  | HBridge.setMotorLeft(0) |
|  | HBridge.setMotorRight(0) |
|  | HBridge.exit() |
|  | print("Program Ended") |
|  | break |
|  |  |
|  | # The keyboard character variable char has to be set blank. We need |
|  | # to set it blank to save the next key pressed by the user. |
|  | char = "" |
|  | # End |

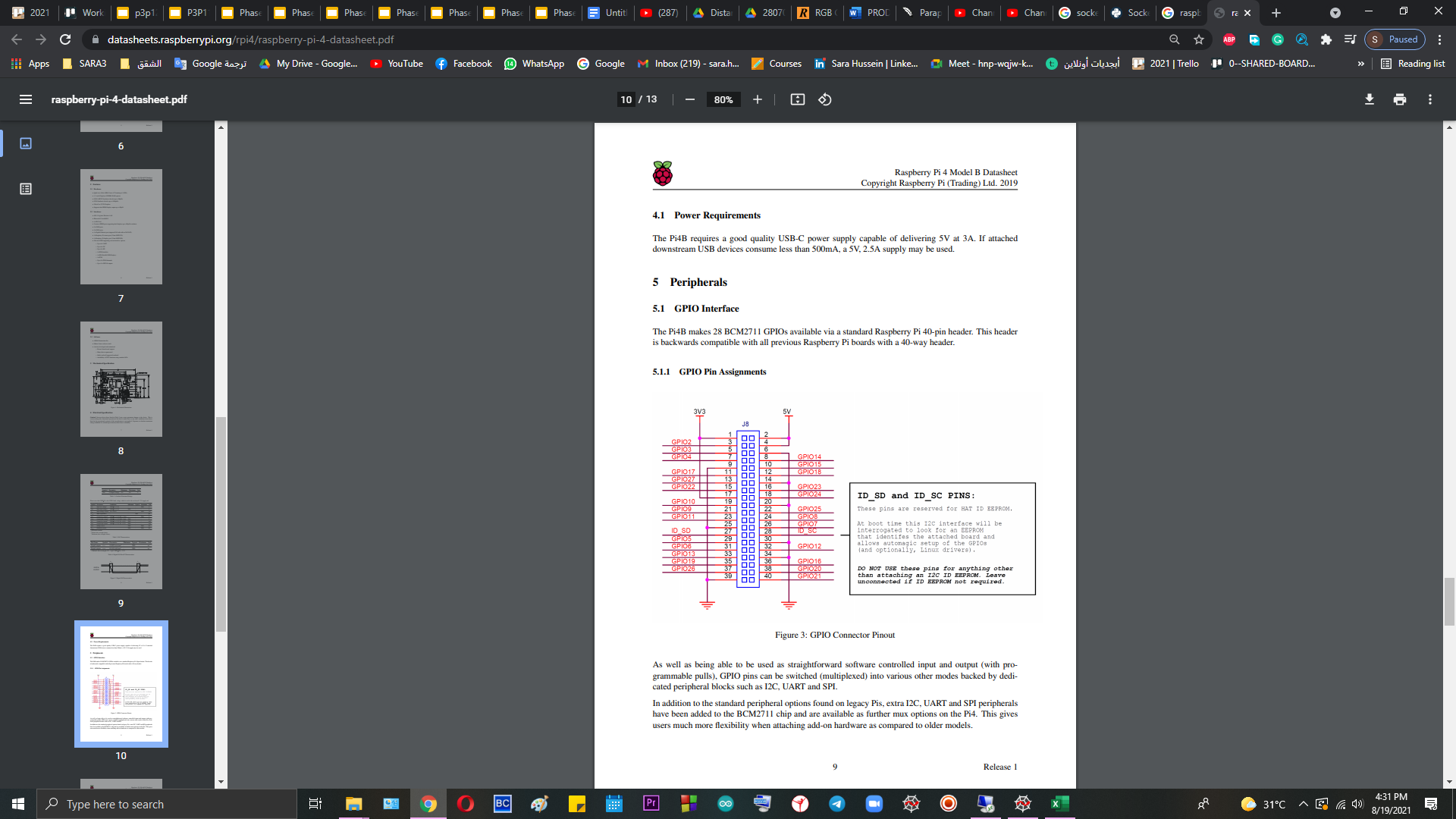
## Appendix C: Data Sheets

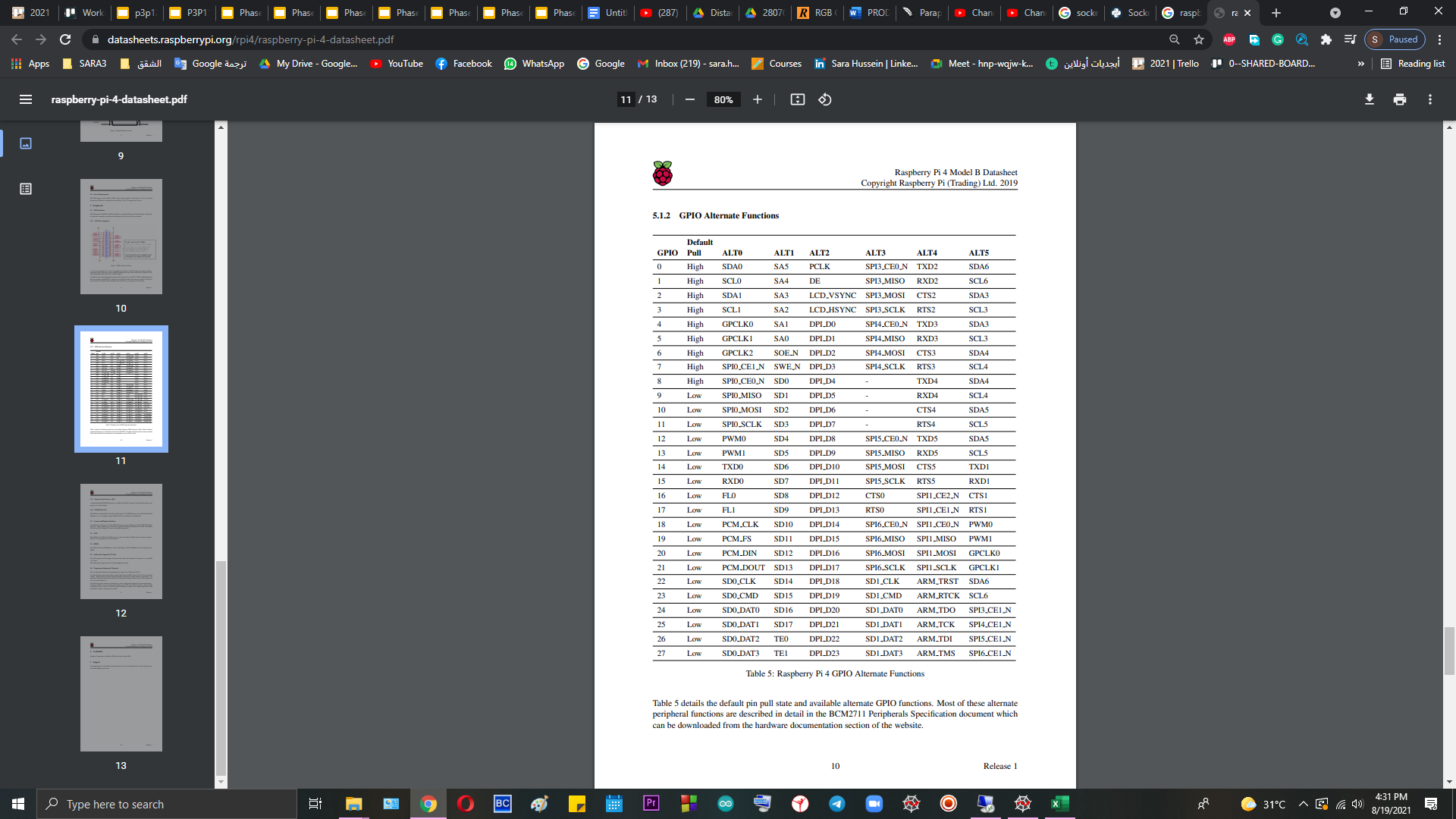
**Raspberry Pi 4 Model B**

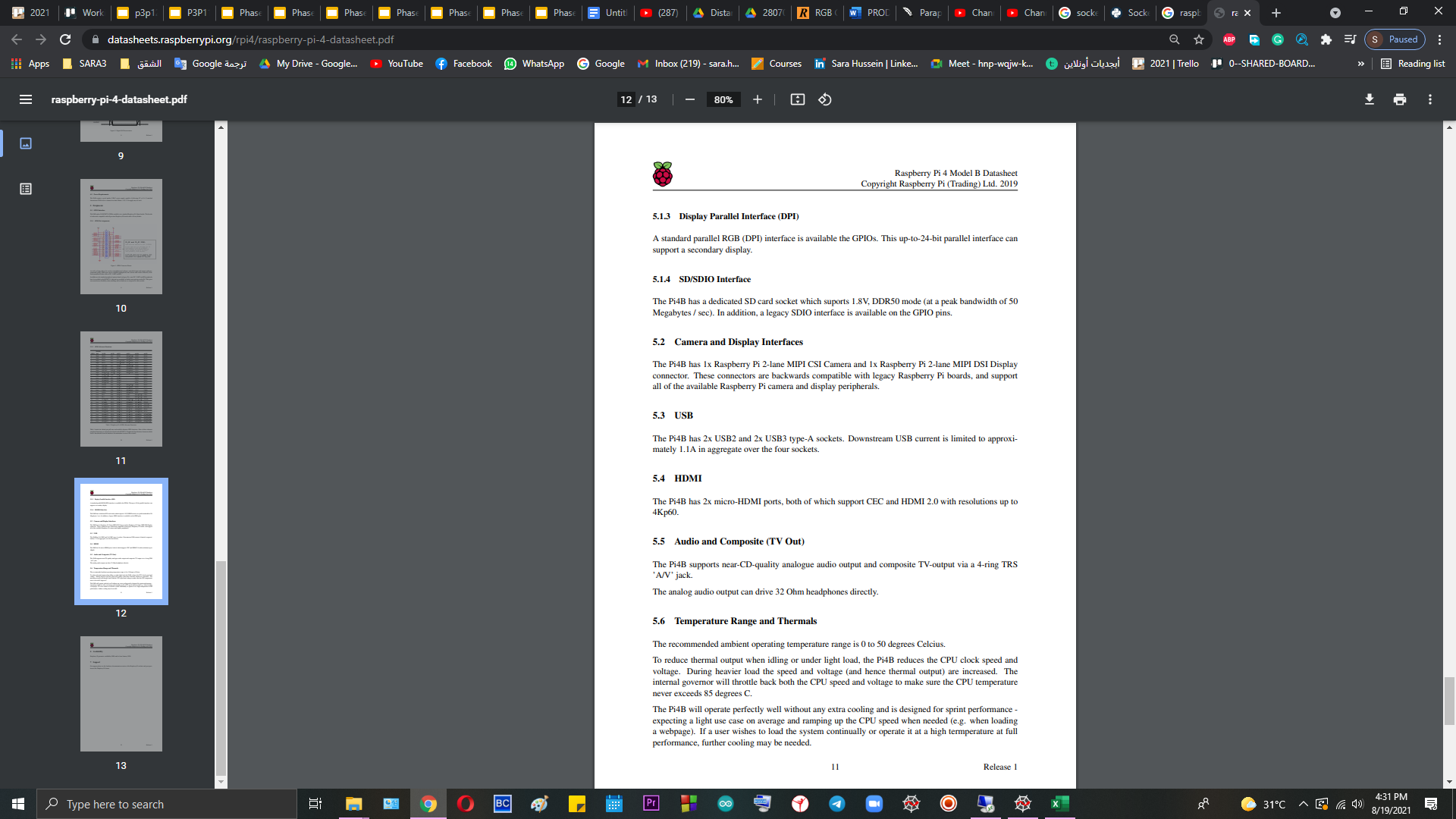


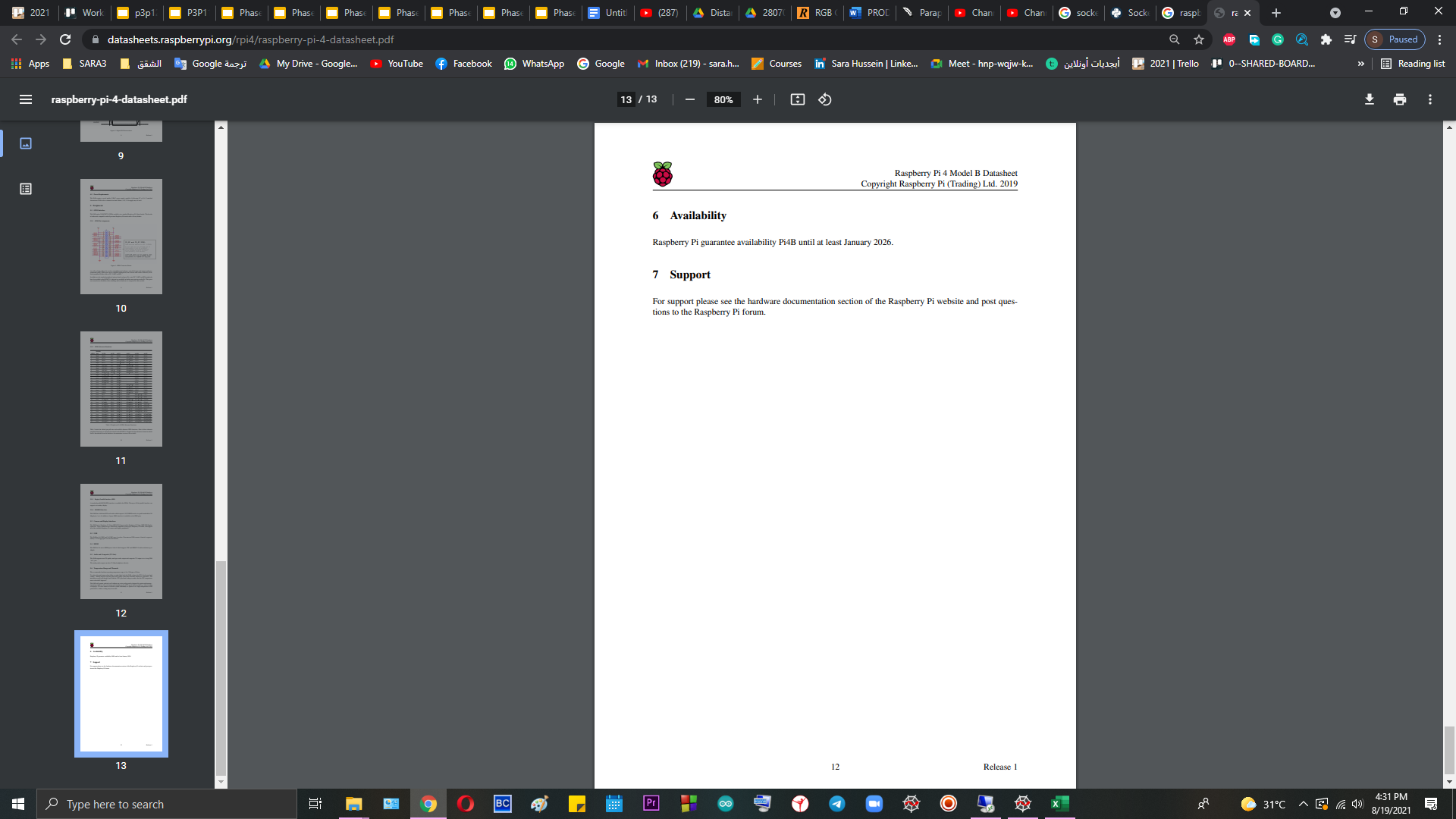




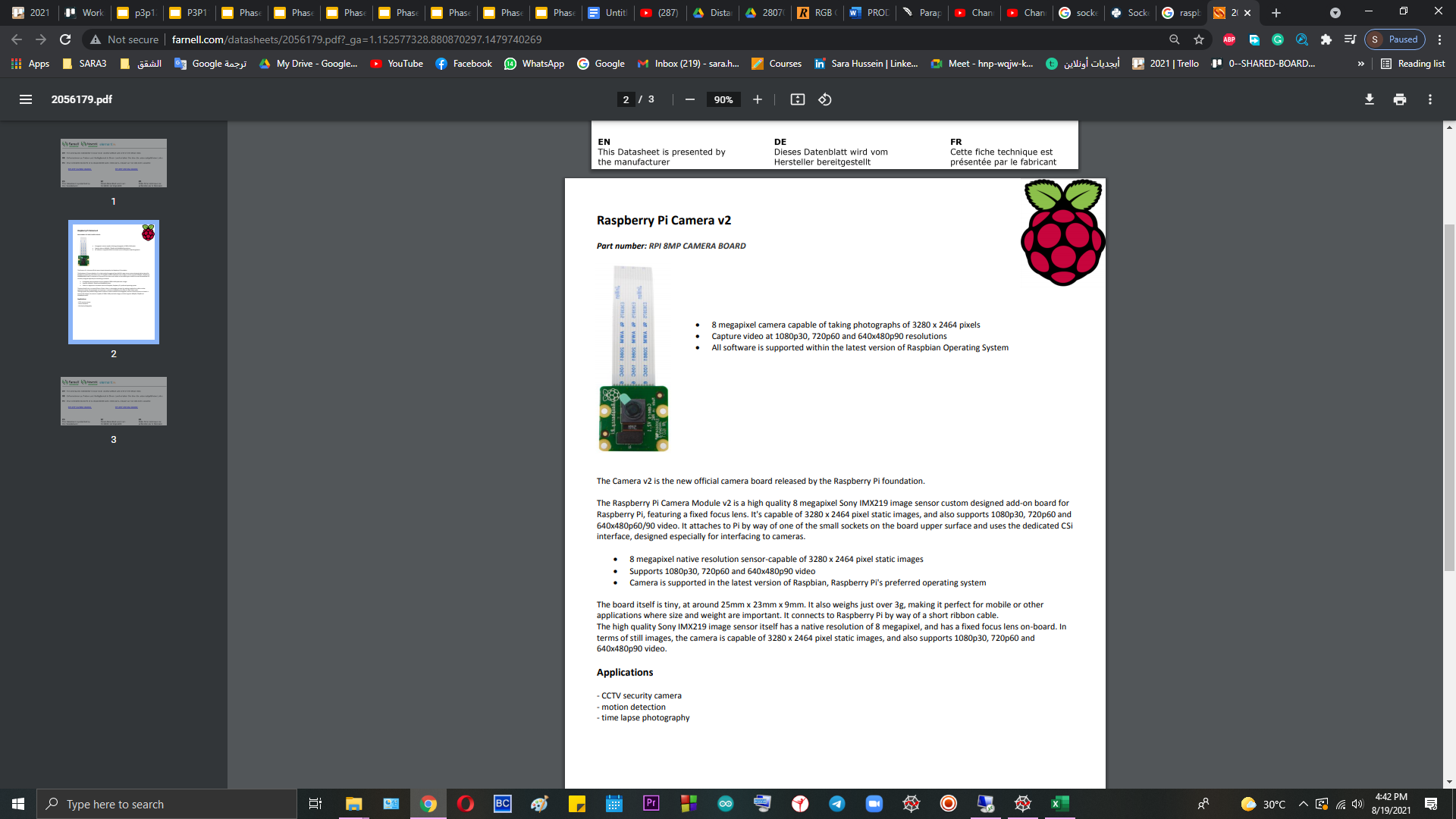








**Raspberry Pi Camera V2**



**MDD10A**

<https://docs.google.com/document/d/1ol8nICCTTw5dAHHE_hju08cCVH2GN5_Y3cGC6B4Gbas/edit>

**LiPo Rechargeable Battery 22.2V 2800mAh 25C**

<https://my.cytron.io/p-h-42483?search=LiPo%20Rechargeable%20Battery%2022.2V%202800mAh%2025C&description=1>

**24V 148RPM 18kgfcm 45mm Planetary DC Geared Motor**

<https://drive.google.com/file/d/0BzFWfMiqqjyqcGFkdXpYa21FSVU/view?resourcekey=0-5y1Be9zDfju-R6Fvs0KJSw>

<https://drive.google.com/file/d/1zFnEdazvZiHd57JwrJtNiY_-n2of-lRk/view>

**Ultrasonic Ranging Module HC - SR04**

<https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>

<https://my.cytron.io/p-5v-hc-sr04-ultrasonic-sensor?search=hc-sr04&description=1>

**Infrared Sensor Module**

<https://my.cytron.io/p-infrared-sensor-module?search=i%20sensor&description=1>