**Dia-Bot: Installation Diagnostic Robot**

Text

Description automatically generated with low confidence

**High-Fidelity Prototype**

**Fabrication Package**

Bill of Materials – Electrical

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dia-Bot Electrical Bill of Materials** |  |  |  |  |
| **Name** | **Part Number** | **Cost (Per)** | **Quantity** | **Link** |
| Raspberry Pi 4 | n/a | $ 35.00 | 1 | [Buy a Raspberry Pi 4 Model B – Raspberry Pi](https://www.raspberrypi.com/products/raspberry-pi-4-model-b/) |
| Raspberry Pi Spy Camea | n/a | $ 39.95 | 1 | [Spy Camera for Raspberry Pi : ID 1937 : $39.95 : Adafruit Industries, Unique & fun DIY electronics and kits](https://www.adafruit.com/product/1937) |
| 6DF Accel/Gyro | MPU-6050 | $ 6.95 | 3 | [Adafruit MPU-6050 6-DoF Accel and Gyro Sensor - STEMMA QT Qwiic : ID 3886 : $6.95 : Adafruit Industries, Unique & fun DIY electronics and kits](https://www.adafruit.com/product/3886) |
| DC Motors (Drive) | AM-4230 | $ 45.00 | 2 | <https://www.andymark.com/products/johnson-electric-gearmotor-and-output-shaft> |
| Servo Motors (Camera) | SER0043 | $ 4.00 | 2 | [SER0043 DFRobot | Motors, Solenoids, Driver Boards/Modules | DigiKey](https://www.digikey.com/en/products/detail/dfrobot/SER0043/7087197) |
| H-Bridge Driver | L9110H | $ 1.50 | 2 | <https://www.adafruit.com/product/4489> |
| H-Bridge Breakout | TB6612 | $ 4.95 | 1 | <https://www.adafruit.com/product/2448> |
| Temperature sensor | MAX31820 | $ 1.95 | 1 | <https://www.sparkfun.com/products/14049> |
| Microphone (Electret) | MAX4466 | $ 6.95 | 1 | <https://www.sparkfun.com/products/12758> |
| Buck Converter: 18V-5V | MPM3610 | $ 9.95 | 1 | <https://www.sparkfun.com/products/18375> |
| Buck Converter: 18V-12V | WG8-40S1203 | $ 15.99 | 1 | [Amazon.com: DC Voltage Reducer Converter DC 8V-40V to 12V 3A 36W Automatic Step Down Up Voltage Regulator Power Converter Waterproof Module Transformer for Golf Cart Club Car : Electronics](https://www.amazon.com/gp/product/B07WY4P7W8/ref=ppx_od_dt_b_asin_title_s00?ie=UTF8&th=1) |
| Milwaukee 18V Battery | M18 | $ 30.99 | 1 | [For Milwaukee 18V Battery Replacement | M18 3.0Ah Li-Ion Battery — Vanon-Batteries-Store (vanonbatteries.com)](https://www.vanonbatteries.com/products/for-milwaukee-18v-battery-replacement-48-11-1850-3-0ah-li-ion-battery?variant=49010308420&msclkid=41b41373152713eed399720edce809e9&utm_source=bing&utm_medium=cpc&utm_campaign=%E6%A0%87%E5%87%86%E8%B4%AD%E7%89%A9-2020.11.18&utm_term=4588880498903728&utm_content=Ad%20group%20%231) |
| White LEDs | COM-11121 | $ 0.55 | 5 | [Diffused LED - White 10mm - COM-11121 - SparkFun Electronics](https://www.sparkfun.com/products/11121) |

Bill of Material – Mechanical

Diagram

Description automatically generated

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Dia-Bot Mechancial Bill of Materials** |  |  |  |  |  |
| **Name** | **ID** | **Part Number** | **Cost (Per)** | **Quantity** | **Link** |
| 1045 Carbon Steel Keyed Rotary Shaft, 10x300mm | 1 | |  | | --- | | 1439K311 | | $24.21 | 2 | <https://www.mcmaster.com/rotary-shafts/keyed-rotary-shafts-5/system-of-measurement~metric/> |
| Bronze Sleeve Bearings, 10mm | 2 | 2938T763 | $2.19 | 24 | <https://www.mcmaster.com/bushings/bearings-3/high-load-oil-embedded-flanged-sleeve-bearings/system-of-measurement~metric/> |
| Shaft Collar, 10mm | 3 | G0318906 | $2.73 | 24 | <https://www.zoro.com/ruland-manufacturing-shaft-collar-set-screw-1pc-10mm-steel-msc-10-f/i/G0318906/> |
| Thick Aluminum Composite ACM Black Sheet 24in x 72in | 4 |  | $65.67 | 1 | <https://www.homedepot.com/p/Falken-Design-24-in-x-72-in-x-1-8-in-Thick-Aluminum-Composite-ACM-Black-Sheet-Falken-Design-ACM-BK-1-8-2472/308670300?MERCH=REC-_-pip_alternatives-_-308670312-_-308670300-_-N&> |
| M4 x 60mm Female Threaded Brass Hex Standoff Pillar Spacer Nut 10pcs | 5 |  | $9.99 | 1 | <https://www.amazon.com/Female-Threaded-Standoff-Pillar-Spacer/dp/B0177VG4Q8> |
| Yeah Racing 90mm Desert Lizard Two Stage Internal Spring Shock |  | YEA-DDL-090RD | $24.99 | 4 | <https://www.rcplanet.com/yeah-racing-90mm-desert-lizard-two-stage-internal-spring-shock-2-red-yea-ddl-090rd/p-tuqesxwqtm2xweue> |
| Brecoflex Self-Tracking Timing Belt Two Pack |  | 50 TK5K6/1100 V | $188.94 | 1 | <https://www.brecoflex.com/products/timing-belts/self-tracking-series/> |
| 3mmx3mm Machine Key Stock |  | 92288A715 | $4.14 | 1 | <https://www.mcmaster.com/machine-keys/machine-key-stock-5/system-of-measurement~metric/> |
| ½” Roller Bearing | 6 | 25015T24 | $7.24 | 8 | <https://www.mcmaster.com/bearings/wheel-axles-bearings-and-reducer-bushings/> |
| Low-Profile Sleeve Bearing Carriagefor 27 mm Wide Rail |  | 6723K11 | 6.77 | 4 | <https://www.mcmaster.com/6723K11/> |
| 27 mm Wide Guide Rail for Low-Profile Sleeve Bearing Carriage |  | 6723K2 | 17.50 | 2 | <https://www.mcmaster.com/6723K2/> |

Engineering Drawings – Mechanical

Diagram, engineering drawing

Description automatically generated

**Engineering Drawing for Custom 30 Tooth Idle Pulley**

**Diagram, engineering drawing

Description automatically generated**

**Engineering Drawing for Custom 50 Tooth Drive Pulley**

**Diagram, engineering drawing

Description automatically generated**

**Engineering Drawing for Custom Chassis Frame**

Raspberry Pi Code (Python)

Robot\_test.py – main file

import sys

import tkinter as tk

from tkinter import \*

from PIL import ImageTk, Image

import picamera

import time

import threading

import math

from random import \*

import RPi.GPIO as GPIO

import pigpio

import DCMotor

import DualHBridge

import DataCollection

import matplotlib

matplotlib.use("TkAgg")

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk

from matplotlib.figure import Figure

# Initialize necessary variables

camera = picamera.PiCamera()

top = tk.Tk()

top.title('Dia-Bot')

led = 11

pwmPinA = 12

motorAIn1 = 15

motorAIn2 = 16

pwmPinB = 19

motorBIn1 = 21

motorBIn2 = 22

motorEn = 18

speed = IntVar()

zoom = IntVar()

gpioMode = GPIO.BOARD

GPIO.setwarnings(False)

GPIO.setmode(gpioMode)

GPIO.setup(led, GPIO.OUT)

pi = pigpio.pi()

motors = DualHBridge.DualHBridge(pwmPinA, motorAIn1, motorAIn2, pwmPinB, motorBIn1, motorBIn2, motorEn, gpioMode)

# Threading control

graphRefreshTime = 5 # Number of seconds between graph refresh

programRunning = True

collectData = False

uiMutex = threading.Lock()

startTime = time.time\_ns()

# Closes relevant processes and stops GPIO

def exit():

#camera.stop\_preview()

#camera.close()

top.destroy

pwm.stop()

GPIO.output(pwm, GPIO.LOW)

GPIO.cleanup()

quit()

if not pi.connected:

print("Error: Pi Not connected")

exit()

# Debugging function - run a function and report how long it takes

def elapsedTime(func, \*args):

startTime = time.time\_ns()

#try:

# func(\*args)

#except:

func()

elapsedTimeNs = time.time\_ns() - startTime

print("ElapsedTime (" + str(func.\_\_name\_\_) + ") = " + str(elapsedTimeNs / 1\_000\_000) + " ms")

# Returns a printout of the total time since execution began

def totalElapsedTime():

global startTime

return f"(total time = {(time.time\_ns()-startTime)/1\_000\_000\_000} s)"

# Opens the camera preview on the screen

# Note: for VNC users to see the feed, the setting "Enable Direct Capture Mode" must be on

#def start\_camera():

# camera.preview\_fullscreen=False

# camera.preview\_window=(90,100, 1280, 720)

# camera.resolution=(1280,720)

# camera.start\_preview()

# -------- Define UI button functions --------

# Callback function for the zoom scroll bar

def setSpeed(var):

speed = var

def moveForwardPress(event):

print("Moving forward! Press - Speed = " + str(speed.get()))

def moveForwardRelease(event):

print("Release moving forward")

def moveForwardRightPress(event):

print("Moving forward-right! Press - Speed = " + str(speed.get()))

def moveForwardRightRelease(event):

print("Release moving forward-right")

def moveForwardLeftPress(event):

print("Moving forward-left! Press - Speed = " + str(speed.get()))

def moveForwardLeftRelease(event):

print("Release moving forward-left")

def moveBackwardPress(event):

print("Moving backward! Press - Speed = " + str(speed.get()))

def moveBackwardRelease(event):

print("Release moving backward")

def moveBackwardRightPress(event):

print("Moving backward-right! Press - Speed = " + str(speed.get()))

def moveBackwardRightRelease(event):

print("Release moving backward-right")

def moveBackwardLeftPress(event):

print("Moving backward-left! Press - Speed = " + str(speed.get()))

def moveBackwardLeftRelease(event):

print("Release moving backward-left")

def moveLeftPress(event):

print("Turn left! Press")

def moveLeftRelease(event):

print("Release moving left")

def moveRightPress(event):

print("Turn right! Press")

def moveRightRelease(event):

print("Release moving right")

def stopMovement():

print("Emergency stop!")

def lock():

print("Locking suspension")

def ledOn():

print("Turning on LED")

GPIO.output(led, True)

def ledOff():

print("Turning off LED")

GPIO.output(led, False)

# Testing purposes only - to be deprecated

def motorTurnTest():

print("Testing DC motor")

print("What goes up...")

for dc in range(0, 101, 2):

#motor.setVelo(dc)

motors.go(dc)

time.sleep(0.05)

time.sleep(1)

print("...must come down")

for dc in range(100, -1, -2):

#motor.setVelo(dc)

motors.go(dc)

time.sleep(0.05)

time.sleep(1)

print("Aaaand backwards")

for dc in range(0, -101, -2):

#motor.setVelo(dc)

motors.go(dc)

time.sleep(0.05)

print("And back")

for dc in range(-100, 1, 2):

#motor.setVelo(dc)

motors.go(dc)

time.sleep(0.05)

print("Motor turn done")

def cameraUp():

print("Camera tilt up!")

def cameraDown():

print("Camera tilt down!")

def cameraLeft():

print("Camera tilt left!")

def cameraRight():

print("Camera tilt right!")

def takePhoto():

print("Taking photo!")

def soundStatus():

print("Sound data status")

def accelerationStatus():

print("Acceleration data status")

def videoStatus():

print("Video status")

def stopGpio():

GPIO.setmode(gpioMode)

GPIO.output(motorEn, GPIO.LOW)

GPIO.output(pwmPinA, GPIO.LOW)

#pwm.stop()

GPIO.cleanup()

# -------------------------- GUI SETUP CODE --------------------------

top.resizable(width=False, height=False)

top.geometry("1600x900")

# Primary sections

controlFrame = tk.Frame(top, width=400, height=900)#, bg='orange')

dataFrame = tk.Frame(top, width=1120, height=270)#, bg='blue')

videoFrame = tk.Frame(top, width=1120, height=630)#, bg='red')

# Individual Control Frames

movementControls = tk.Frame(controlFrame, width=400, height=280)#, bg='blue')

cameraControls = tk.Frame(controlFrame, width=400, height=280)

alertControls = tk.Frame(controlFrame, width=400, height=280)

# Arrange frames

controlFrame.grid(row=1, column=1, sticky="nesw")

dataFrame.grid(row=1, column=2)

videoFrame.grid(row=2, column=2)

# ------------------ Controls Pane -----------------------

# Controls top text

controlsLabel = tk.Label(controlFrame, text="Controls", font="none 18 bold")

controlsLabel.grid(row=1, column=1, columnspan=8)

controlsLabel.config(anchor=CENTER)

controlFrame.grid\_rowconfigure(1, minsize=60)

# ----- Movement controls -----

movementControls.grid(row=2, column=1, rowspan=2, columnspan=10)

tk.Label(movementControls, text="Movement", anchor=CENTER, font="none 14 bold").grid(row=1, column=1, columnspan=9)

tk.Label(movementControls, text="Speed", anchor=CENTER, font="bold").grid(row=2, column=2)

speedScale = tk.Scale(movementControls, from\_=100, to=0, orient=tk.VERTICAL, variable = speed, length=150, showvalue=0, sliderlength=20)

speedScale.grid(row=3, column=2, rowspan=4)

speedScale.set(50)

# Directional buttons

tk.Label(movementControls, text="Direction", anchor=CENTER, font="bold").grid(row=2, column=4, columnspan=3)

moveForwardButton = tk.Button(movementControls, text="^", anchor=CENTER, font="16")

moveForwardButton.bind("<ButtonPress>", moveForwardPress)

moveForwardButton.bind("<ButtonRelease>", moveForwardRelease)

#top.bind("<KeyPress-w>", moveForwardPress)

#top.bind("<KeyRelease-w>", moveForwardRelease)

moveForwardButton.grid(row=3, column=5)

moveForwardLeftButton = tk.Button(movementControls, text="FL", anchor=CENTER, font="16")

moveForwardLeftButton.bind("<ButtonPress>", moveForwardLeftPress)

moveForwardLeftButton.bind("<ButtonRelease>", moveForwardLeftRelease)

moveForwardLeftButton.grid(row=3, column=4)

moveForwardRightButton = tk.Button(movementControls, text="FR", anchor=CENTER, font="16")

moveForwardRightButton.bind("<ButtonPress>", moveForwardRightPress)

moveForwardRightButton.bind("<ButtonRelease>", moveForwardRightRelease)

moveForwardRightButton.grid(row=3, column=6)

moveBackwardButton = tk.Button(movementControls, text="v", anchor=CENTER, font="16")

moveBackwardButton.bind("<ButtonPress>", moveBackwardPress)

moveBackwardButton.bind("<ButtonRelease>", moveBackwardRelease)

#top.bind("<KeyPress-s>", moveBackwardPress)

#top.bind("<KeyRelease-s>", moveBackwardRelease)

moveBackwardButton.grid(row=5, column=5)

moveBackwardLeftButton = tk.Button(movementControls, text="BL", anchor=CENTER, font="16")

moveBackwardLeftButton.bind("<ButtonPress>", moveBackwardLeftPress)

moveBackwardLeftButton.bind("<ButtonRelease>", moveBackwardLeftRelease)

moveBackwardLeftButton.grid(row=5, column=4)

moveBackwardRightButton = tk.Button(movementControls, text="BR", anchor=CENTER, font="16")

moveBackwardRightButton.bind("<ButtonPress>", moveBackwardRightPress)

moveBackwardRightButton.bind("<ButtonRelease>", moveBackwardRightRelease)

moveBackwardRightButton.grid(row=5, column=6)

moveLeftButton = tk.Button(movementControls, text="<", anchor=CENTER, font="16")

moveLeftButton.bind("<ButtonPress>", moveLeftPress)

moveLeftButton.bind("<ButtonRelease>", moveLeftRelease)

#top.bind("<KeyPress-a>", moveLeftPress)

#top.bind("<KeyRelease-a>", moveLeftRelease)

moveLeftButton.grid(row=4, column=4)

moveRightButton = tk.Button(movementControls, text=">", anchor=CENTER, font="16")

moveRightButton.bind("<ButtonPress>", moveRightPress)

moveRightButton.bind("<ButtonRelease>", moveRightRelease)

moveRightButton.grid(row=4, column=6)

# Stop and lock buttons

tk.Label(movementControls, text="Mode", anchor=CENTER, font="bold").grid(row=2, column=9)

tk.Button(movementControls, text="Stop", command=stopMovement, anchor=CENTER, fg="red", font="16").grid(row=3, column=9)

tk.Button(movementControls, text="Lock", command=lock, anchor=CENTER, font="16").grid(row=5, column=9)

movementControls.grid\_columnconfigure(1, minsize=10)

for i in range(2,10):

movementControls.grid\_columnconfigure(i, minsize=20)

# Keyboard Buttons

top.bind("<KeyPress-w>", moveForwardPress)

top.bind("<KeyRelease-w>", moveForwardRelease)

top.bind("<KeyPress-s>", moveBackwardPress)

top.bind("<KeyRelease-s>", moveBackwardRelease)

top.bind("<KeyPress-a>", moveLeftPress)

top.bind("<KeyRelease-a>", moveLeftRelease)

top.bind("<KeyPress-d>", moveRightPress)

top.bind("<KeyRelease-d>", moveRightRelease)

# ----- Camera Controls -----

cameraControls.grid(row=5, column=1, rowspan=1, columnspan=10)

tk.Label(cameraControls, text="Camera", anchor=CENTER, font="none 14 bold").grid(row=1, column=1, columnspan=9)

# Directional buttons

tk.Label(cameraControls, text="Angle", anchor=CENTER, font="bold").grid(row=2, column=2, columnspan=3)

tk.Button(cameraControls, text="^", command=cameraUp, anchor=CENTER, font="16").grid(row=3, column=3)

tk.Button(cameraControls, text="v", command=cameraDown, anchor=CENTER, font="16").grid(row=5, column=3)

tk.Button(cameraControls, text="<", command=cameraLeft, anchor=CENTER, font="16").grid(row=4, column=2)

tk.Button(cameraControls, text=">", command=cameraRight, anchor=CENTER, font="16").grid(row=4, column=4)

tk.Button(cameraControls, text="Pic", command=takePhoto, anchor=CENTER, font="16").grid(row=4, column=3)

# Stop and lock buttons

tk.Label(cameraControls, text="Light", anchor=CENTER, font="bold").grid(row=2, column=6)

tk.Button(cameraControls, text="On", command=ledOn, anchor=CENTER, font="16").grid(row=4, column=6)

tk.Button(cameraControls, text="Off", command=ledOff, anchor=CENTER, font="16").grid(row=5, column=6)

tk.Label(cameraControls, text="Zoom", anchor=CENTER, font="bold").grid(row=2, column=8)

zoomScale = tk.Scale(cameraControls, from\_=100, to=0, orient=tk.VERTICAL, variable = zoom, length=150, showvalue=0, sliderlength=20)

zoomScale.grid(row=3, column=8, rowspan=4)

zoomScale.set(50)

cameraControls.grid\_columnconfigure(1, minsize=10)

for i in range(2,10):

cameraControls.grid\_columnconfigure(i, minsize=20)

# Alerts

alertControls.grid(row=7, column=1, rowspan=1, columnspan=10)

tk.Label(alertControls, text="Alerts", anchor=CENTER, font="none 14 bold").grid(row=1, column=1, columnspan=9)

tk.Label(alertControls, text="Active", anchor=CENTER, font="none 11").grid(row=2, column=4, columnspan=2)

tk.Label(alertControls, text="Threshold", anchor=CENTER, font="none 11").grid(row=2, column=7, columnspan=2)

tk.Label(alertControls, text="Alerts", anchor=CENTER, font="none 11").grid(row=2, column=10, columnspan=2)

nextRow = 3

# Repeated function to add new alert categories to UI

def addAlert(name, thresholdUnits):

global nextRow

#print("Adding alert row for " + name)

tk.Label(alertControls, text=name, anchor="w", justify=LEFT, font="none 11").grid(row=nextRow, column=1, columnspan=3)

tk.Checkbutton(alertControls, text="Enabled", anchor=CENTER, font="none 11").grid(row=nextRow, column=4, columnspan=2)

tk.Entry(alertControls, justify=CENTER, width=5, font="none 11").grid(row=nextRow, column=6, columnspan=2)

tk.Label(alertControls, text=thresholdUnits, anchor="w", justify=LEFT, font="none 11").grid(row=nextRow, column=8, columnspan=2)

if randint(0,1) == 1:

tk.Label(alertControls, text="Error", anchor=CENTER, font="none 11 bold", fg="red").grid(row=nextRow, column=10, columnspan=2)

else:

tk.Label(alertControls, text="None", anchor=CENTER, font="none 11", fg="black").grid(row=nextRow, column=10, columnspan=2)

nextRow = nextRow + 1

# Call above function to add alerts to UI

addAlert("Vibration", "m/s2")

addAlert("Sound", "dB")

addAlert("Temperature", "°C")

alertControls.grid\_columnconfigure(1, minsize=10)

for i in range(2,10):

alertControls.grid\_columnconfigure(i, minsize=20)

for i in range(2, nextRow+1):

alertControls.grid\_rowconfigure(i, minsize=30)

# ----- Other -----

# Testing buttons

tk.Button(controlFrame, text="Motor", command=motorTurnTest).grid(row=14, column=2)

tk.Button(controlFrame, text="Off", command=stopGpio).grid(row=14, column=3)

def toggleData():

global collectData

collectData = not collectData

print("Setting colletData to " + str(collectData))

# ------------------ Data Pane -----------------------

tk.Label(dataFrame, text="Data", font="none 18 bold").grid(row=1, column=1, columnspan=50)

tk.Button(dataFrame, text="Toggle Data", command=toggleData).grid(row=1, column=2)

# Individual Frames

soundLevelFrame = tk.Frame(dataFrame, width=350, height=350)#, bg="red")

vibrationFrame = tk.Frame(dataFrame, width=350, height=350)#, bg="yellow")

temperatureFrame = tk.Frame(dataFrame, width=350, height=350)#, bg="orange")

positionFrame = tk.Frame(dataFrame, width=350, height=350)#, bg="green")

dataFrames = [soundLevelFrame, vibrationFrame, temperatureFrame, positionFrame]

units = ["dB", "m/s2", "C", "m"]

# Sound Level

def readSoundLevel():

num = randint(-10, 10)

#print("Reading sound level! - " + str(num))

return num

soundLevel = DataCollection.DataCollection("Sound Level", "dB", soundLevelFrame, readSoundLevel, uiMutex, startTime)

soundLevel.tkAddDataPane()

soundLevelFrame.grid(row=2, column=1, padx=10)

# Vibration

def readVibration():

num = randint(-10, 10)

#print("Reading vibration! - " + str(num))

return num

vibration = DataCollection.DataCollection("Vibration", "m/s2", vibrationFrame, readVibration, uiMutex, startTime)

vibration.tkAddDataPane()

vibrationFrame.grid(row=2, column=2, padx=10)

# Temperature

def readTemperature():

num = randint(-10, 10)

#print("Reading temperature! - " + str(num))

return num

temperature = DataCollection.DataCollection("Temperature", "°C", temperatureFrame, readTemperature, uiMutex, startTime)

temperature.tkAddDataPane()

temperatureFrame.grid(row=2, column=3, padx=10)

# Position

def readPosition():

num = randint(-10, 10)

#print("Reading position! - " + str(num))

return num

position = DataCollection.DataCollection("Position", "m", positionFrame, readPosition, uiMutex, startTime)

position.tkAddDataPane()

positionFrame.grid(row=2, column=4, padx=10)

# Group of all the data classes

dataClassList = [soundLevel, vibration, temperature, position]

def addDataPress(event):

x = max(x1) + 1

y = randint(-5, 8)

x1.append(x)

y1.append(y)

plot1.plot(x1, y1)

canvas.draw()

print("Adding data! Press - x = " + str(x) + ", y = " + str(y))

def addDataRelease(event):

x = max(x1) + 1

y = randint(-5, 8)

x1.append(x)

y1.append(y)

plot1.plot(x1, y1)

canvas.draw()

print(f"Adding data! Release - x = {x}, y = {y}")

addDataButton.bind("<ButtonPress>", addDataPress)

addDataButton.bind("<ButtonRelease>", addDataRelease)

# ------------------ Video Pane -----------------------

#tk.Button(videoFrame, text="Video", command=videoStatus).grid(row=1, column=1)

testImg = ImageTk.PhotoImage(Image.open("vanderlandeTest.png"))#.resize((1000, 500)))

imgLabel = Label(videoFrame, image=testImg)

imgLabel.grid(row=1, column=1)

def updateFrameImage():

fileName = "frame.png"

camera.capture(fileName)

img = ImageTk.PhotoImage(Image.open(fileName).resize((1000, 500)))

imgLabel = Label(videoFrame, image=img)

imgLabel.grid(row=1, column=1)

# Place the frames

controlFrame.place(relx=0.01, rely=0.01, anchor=tk.NW)

dataFrame.place(relx=0.3, rely=0.01, anchor=tk.NW)

videoFrame.place(x=400, y=300, anchor=tk.NW)

# ----- Threading functions -----

# Wrapper to other functions which loops

def loopAtFrequency(freqHz, loopFunction, \*args):

global programRunning

loopTime = 1/freqHz

while programRunning:

loopStartTime = time.time()

loopFunction(\*args)

loopEndTime = time.time()

loopTimeTaken = loopEndTime - loopStartTime

timeRemaining = loopTime - (loopTimeTaken)

if timeRemaining > 0:

time.sleep(timeRemaining)

else:

print(f"Thread {loopFunction.\_\_name\_\_} took longer to execute ({loopTimeTaken} s) than its given time({loopTimeTaken} s)! Assigning 1s sleep")

time.sleep(1)

# Every data class needs it own graph, data set, and sensor function

def updateData():

global collectData

if collectData:

for dataClass in dataClassList:

dataClass.readData()

#tk.Button(videoFrame, text="Update Image", command=updateFrameImage).grid(row=2, column=1)

def updateGraphs():

global collectData

if collectData:

try:

top.event\_generate("<<graphEvent>>")

except:

print("Error in event\_generate! Unable to update graphs")

def updateGraphsWrapper(event):

elapsedTime(updateGraphsHandler)

def updateGraphsHandler():

global startTime

global loopCount

for dataClass in dataClassList:

dataClass.updateGraph()

# Threading version

#threads = []

#for dataClass in dataClassList:

# print(f"Start threads: {dataClass.name} Loop #{loopCount} {totalElapsedTime()}")

# newThread = threading.Thread(target=dataClass.updateGraph, args=(loopCount,))

# newThread.start()

# threads.append(newThread)

#print(f"Started threads Loop #{loopCount} {totalElapsedTime()} ")

#print("Join threads.. " + str(threads))

#for t in threads:

# t.join()

#print("Completed joining")

top.bind("<<graphEvent>>", updateGraphsWrapper)

def printTime(\*args):

print(totalElapsedTime())

def main():

global programRunning

global startTime

# Create additional threads

dataThread = threading.Thread(target=loopAtFrequency, args=(1, updateData))

graphThread = threading.Thread(target=loopAtFrequency, args=(1/graphRefreshTime, updateGraphs))

# Start threads

dataThread.start()

graphThread.start()

#totalTimeNs = time.time\_ns() - startTime

#print("Start thread time: " + str((totalTimeNs/1\_000\_000)) + " ms")

top.mainloop()

programRunning = False # Stop extra threads

if \_\_name\_\_ == "\_\_main\_\_":

main()

DataCollection.py – Class used to support each type of data

import sys

import tkinter as tk

from tkinter import \*

from PIL import ImageTk, Image

import time

import threading

import math

from random import \*

import RPi.GPIO as GPIO

import pigpio

import picamera

import DCMotor

import DualHBridge

import matplotlib

matplotlib.use("TkAgg")

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg, NavigationToolbar2Tk

from matplotlib.figure import Figure

# GPIO Startup

gpioMode = GPIO.BOARD

GPIO.setwarnings(False)

GPIO.setmode(gpioMode)

pi = pigpio.pi()

class DataCollection:

def \_\_init\_\_(self, name, units, tkTop, readDataFunction, globalUImutex, globalStartTime):

self.name = name

self.units = units

self.tkTop = tkTop

self.readDataFunction = readDataFunction

self.data = []

self.dataMutex = threading.Lock()

self.globalUImutex = globalUImutex

self.globalStartTime = globalStartTime

# Create Random Graph

self.t = list(range(10))

self.data = [randint(-5, 8) for i in range(10)]

self.fig = Figure(figsize=(3,2.5), dpi=80)

self.fig.patch.set\_facecolor("#DBDBDB")

self.plot1 = self.fig.add\_subplot(111)

self.plot1.plot(self.t, self.data)

self.plot1.set\_ylabel(self.units)

def tkAddDataPane(self):

# Top label

tk.Label(self.tkTop, text=self.name, font="none 12 bold").grid(row=1, column=1, columnspan=5)

# Add random graph

self.canvas = FigureCanvasTkAgg(self.fig, master=self.tkTop)

self.canvas.draw()

self.canvas.get\_tk\_widget().grid(row=2, column=1, rowspan=3, columnspan=4)

def readData(self):

self.dataMutex.acquire()

self.t.append(self.t[-1]+1)

self.data.append(self.readDataFunction())

self.dataMutex.release()

def updateGraph(self):

start = 0

end = self.t[-1]

if end > 20:

start = end-20

self.plot1.cla() # Find way to update graphs without it taking too long!!

print(f"Update {self.name} graph: indices ({start}..{end})")

# Update plot

self.dataMutex.acquire()

self.plot1.plot(self.t[start:end], self.data[start:end])

self.dataMutex.release()

# Update canvas on UI

#self.globalUImutex.acquire()

self.canvas.draw()

print(f"Graph done: {self.name} Loop #{loopCount} (total time = {(time.time\_ns()-self.globalStartTime)/1\_000\_000\_000} s)")

#self.globalUImutex.release()

DualHBridge.py – Class to interface with a Dual H-Bridge breakout controlling two motors

import time

import math

import RPi.GPIO as GPIO

import pigpio

import DCMotor

class DualHBridge:

def \_\_init\_\_(self, pwmA, in1A, in2A, pwmB, in1B, in2B, en, gpioMode = GPIO.BOARD):

self.gpioMode = gpioMode

self.motorA = DCMotor.DCMotor(pwmA, in1A, in2A, en, gpioMode)

self.motorB = DCMotor.DCMotor(pwmB, in1B, in2B, en, gpioMode)

def brake(self):

self.motorA.brake()

self.motorB.brake()

def go(self, velo):

self.motorA.setVelo(velo)

self.motorB.setVelo(velo)

def stop(self):

self.motorA.stop()

self.motorB.stop()

DCMotor.py – class to control individual DC motors

import time

import math

import RPi.GPIO as GPIO

import pigpio

class DCMotor:

def \_\_init\_\_(self, pwmPin, in1, in2, en, gpioMode = GPIO.BOARD):

# Initialize variables

self.pwmPin = pwmPin

self.in1 = in1

self.in2 = in2

self.en = en

self.gpioMode = gpioMode

# Initialize GPIO pins

self.gpioStart()

GPIO.output(pwmPin, GPIO.LOW)

self.pwm = GPIO.PWM(pwmPin, 1000) # Frequency=1KHz

self.pwm.start(0)

def gpioStart(self):

# Initialize GPIO pins

GPIO.setwarnings(False)

GPIO.setmode(self.gpioMode)

GPIO.setup(self.in1, GPIO.OUT)

GPIO.setup(self.in2, GPIO.OUT)

GPIO.setup(self.en, GPIO.OUT)

GPIO.setup(self.pwmPin, GPIO.OUT)

def brake(self):

self.gpioStart()

GPIO.setmode(self.gpioMode)

GPIO.output(self.pwmPin, GPIO.LOW)

self.pwm.ChangeDutyCycle(0)

GPIO.output(self.in1, GPIO.HIGH)

GPIO.output(self.in2, GPIO.HIGH)

def stop(self):

self.gpioStart()

GPIO.setmode(self.gpioMode)

GPIO.output(self.en, GPIO.LOW)

GPIO.output(self.pwmPin, GPIO.LOW)

self.pwm.stop()

GPIO.cleanup()

# Velo [-100, 100]

def setVelo(self, velo):

self.gpioStart()

GPIO.setmode(self.gpioMode)

# Sets relative motor speed and direction

if (velo == 1000):

# Brake if the speed is zero

self.brake()

else:

GPIO.output(self.en, GPIO.HIGH)

speed = abs(velo) # NOTE: relative speed is capped at 100

if (speed > 100):

speed = 100

# Set direction

if (velo > 0):

# Forwards

GPIO.output(self.in1, GPIO.HIGH)

GPIO.output(self.in2, GPIO.LOW)

else:

# Backwards

GPIO.output(self.in1, GPIO.LOW)

GPIO.output(self.in2, GPIO.HIGH)

self.pwm.ChangeDutyCycle(speed)

#ssself.stop()