Power Wheelchair ROPs

ME 641 | Sponsors: MSAC | KUME Spring 2021

Instr: Dr. Fischer & Prof. DeAgostino | Grace Campbell, Michael Caradine, Jackson Hayes, Brenna Morris, Juan Pablo Ramirez, and Jessica Russell

Introduction

- Sponsor: Multiple Sclerosis Achievement Center (MSAC)
  - Multiple Sclerosis is disease that causes nerve damage, disrupting communication between the brain and the body
  - Members diagnosed with MS are bound to a wheelchair and are vulnerable to risks everyday
- Prior Progress (Spring 2020)
  - Included a Python code, 6 LiDAR sensors, 3 LEDs, buzzer, Raspberry Pi, and a housing compartment
  - Spring 2021 group was not able to pick up where previous group left off

Purpose

- Create a wheelchair roll-off-prevention (ROPs) system for MSAC patients
- Enhance safety features of the wheelchair
  - LED and noise indicators
- Successfully utilize LiDAR sensors and safety features to create a product suitable for commercialization
  - 6 LiDAR sensors
  - LED strip indicators
  - Increase user friendliness

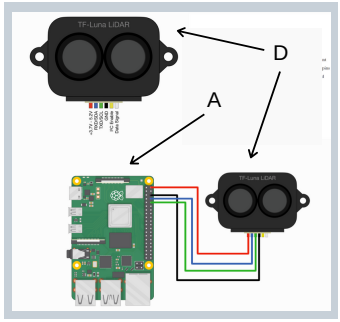
Testing Procedures

- Code & Wiring
  - Found and fixed errors found in the existing code and wire map
- LiDAR Sensors
  - 6 existing sensors troubleshooted individually
  - Successfully able to read distances [m]
- LEDs
  - Green, yellow, and red lights lit up correctly according to distances entered in code

Design

Components:

- A: Raspberry Pi
- B: Big7 USB Expander
- C: USB to Serial Converter
- D: LiDAR Sensors
- E: Speaker
- F: LED Indicator



Results and Analysis

- Edited and fixed existing code
- Successfully incorporated 6 sensors into system
- Tested LED range accuracy
- Able to decrease the loop time in code for sensor readings (.05s)
- Determined Rapsberry Pi B+ optimal for function

Conclusions & Future Work

- Conclusions
  - Will need 8 sensors for full coverage
  - Improved functionality & durability
- Future Work
  - Add stopping mechanism
  - Wire in 360° RGB LEDs around control panel

Acknowledgements

- Judy Markwardt-Oberheu, MSAC
- Professor DeAgostino
- Dr. Kenneth Fischer
- Al Syvongsay
- Thomas Woodruff
- The University of Kansas, School of Engineering

References

[1] 2020 Capstone Group "Final Documentation". 28 April 2020.

[2] TFMini - Micro LiDAR Module Hookup Guide, learn.sparkfun.com/tutorials/tfmini---micro-lidar-module-hookup-guide/all.

[3] Jay_Ali, and Instructables. "Benewake LiDAR TFmini (Complete Guide)." Instructables, Instructables, 4 Mar. 2018, www.instructables.com/Benewake-LiDAR-TFmini-Complete-Guide/.

Power Wheelchair ROPs Experimental Setup Guide

Experimental Setup & Connections

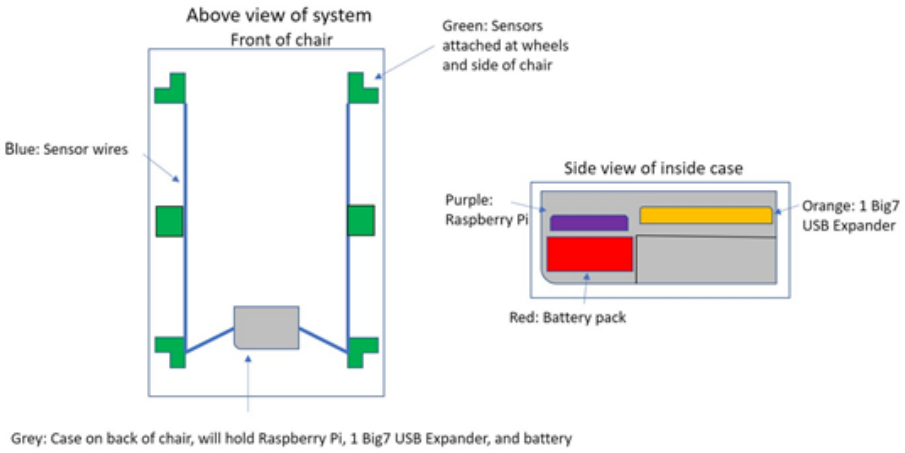
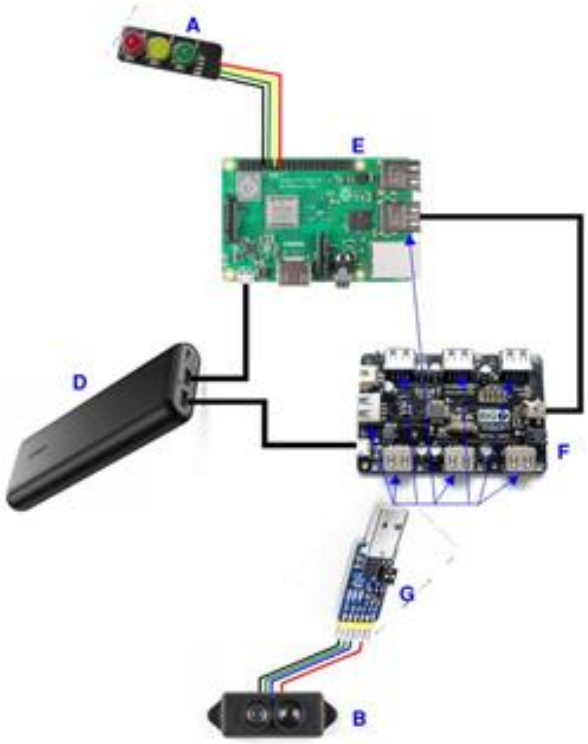
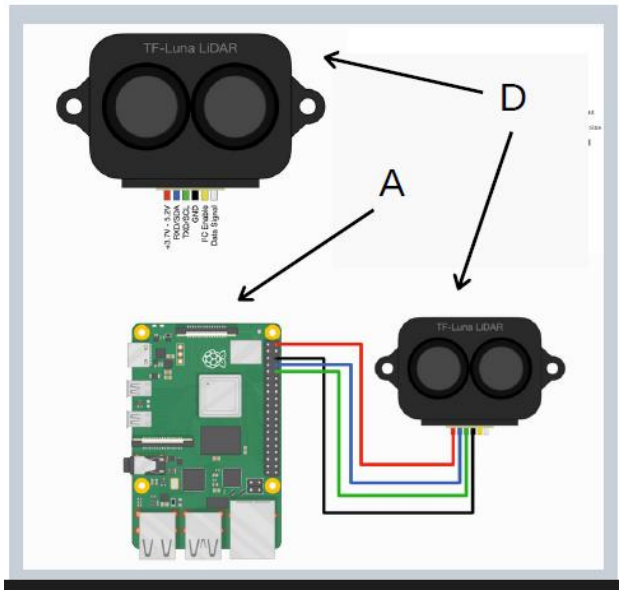


Figure 9: Basic Diagram of System Around Chair and in Case





A: LED Indicator
B: LiDAR Sensors
C: Housing Box
D: Battery
E: Raspberry Pi
F: Big7
G: USB to Serial Converter

Figure 10: Wire Diagram and Outer Components on Wheelchair

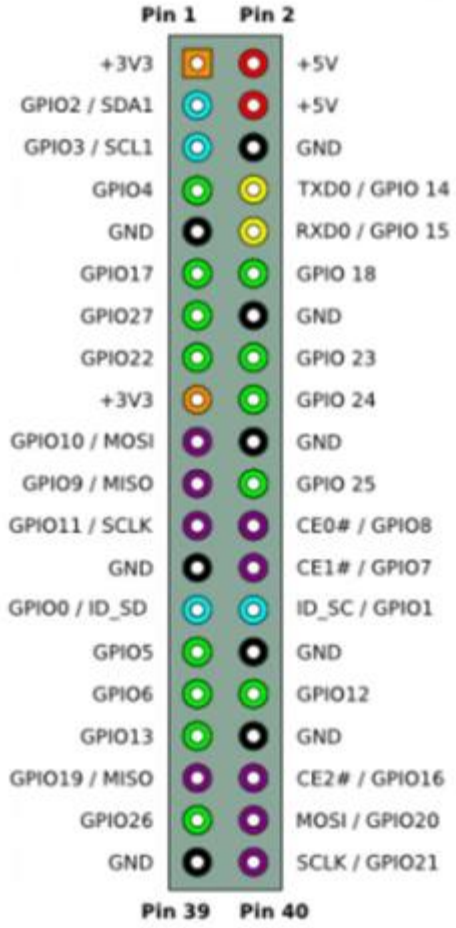


Figure 11: GPIO pinout diagram for Raspberry Pi

Step-by-Step Procedure

1. Make all the proper connections using the connection setup diagram, Figure 9. The order for the wires connecting to USB to serial converter to the Lidar sensor is black, green, blue, red.
    a. Make sure the Big7 and the Raspberry Pi B+ have adequate power by properly connecting the Anker PowerCore Battery.
    b. Be sure to use a monitor, keyboard, and mouse to properly access the Pi software and read the data value readings being displayed.
    c. Connect the Raspberry Pi to the power source last.
    d. Be sure the Raspberry Pi sits securely inside its case. If the underside pins are exposed, it could risk shorting the circuit.
2. After the connections are made, the monitor should show the Pi booting up. It will pull up a software called Thonny where all the python (.py) files are stored on the SD card.

3. The most up-to-date version of the code is the 8working2021.py. This can be opened and edited by going to the file manager in the top left corner, then going to Documents and clicking on the Code subfolder within the Capstone folder.
4. To run this code, go to the top left corner and open the terminal. In the terminal window, enter the commands: “cd Documents”, “cd Capstone”, “cd Code”, “python 8working2021.py”.
    a. If the components connections are correct, the terminal will display readings as Distance0, Distance1, Distance2, Distance3, Distance4, Distance5, Distance6, Distance7. It will also display the sensors labeled as USB0 to USB7 depending on the order which the converters are plugged into the Big7 USB expander.
    b. If the terminal shows an error message, one can verify the Pi is reading all the required devices by running the “ls usb” command in the terminal.
5. After verifying the code is correctly running manually, the next step is to run the code automatically on the Pi as it starts up. Ultimately, the Pi is connected to the monitor and commands are entered into the terminal that tell it the run the script automatically after the Pi reboots. When this works, the Pi should be able to be unplugged from the monitor and into the box attached to the back of the chair to run automatically. There are different methods to do this, the init.d method is likely the best one for this project. *NOTE: The autorun ability on this project still needs further troubleshooting to correct inconsistencies with boot sequence initialization before the script runs. The following instructions explain the init.d method, and with this, further troubleshooting will be needed. *
    a. Using the terminal, add the 8working2021.py file to the init.d directory using the following line
        i. “sudo cp /home/pi/8working2021.py /etc/init.d/”
    b. Move to the init directory and open the script:
        i. Type command “cd /etc/init.d”, *press enter*, type command “sudo nano 8working2021.py” *press enter*.
    c. Enter the following lines to the top of the 8working2021.py script in the init directory:

BEGIN INIT INFO

Provides: 8working2021.py

Required-Start: $all

Required-Stop: $all

Short-Description: run 8working2021.py on boot

END INIT INFO

    d. Make the script in the init directory executable by changing its permission

        i. “sudo chmod +x 8working2021.py”
    e. Run the command:
        i. “sudo update-rc.d 8working2021.py defaults”
    f. Reboot the Pi to see if the script runs automatically
        i. “sudo reboot”
6. When finished using the Pi and turning off the Raspberry Pi, be sure to do so by running the command “sudo shutdown now” in the terminal.
    a. Do not turn off the Pi by unplugging the power source from it. It could possibly break it.

KU Mechanical Engineering: Power Wheelchair Roll-Over Prevention System

Fall 2020 – Spring 2021

Grace Campbell, Michael Caradine, Jackson Hayes, Brenna Morris, Juan Pablo Ramirez, Jessica Russell

Background

Wheelchairs are commonly used by those who need assistance moving around. Furthermore, some people who use these wheelchairs have neurological illnesses that result in reduced motor function. Multiple sclerosis (MS) is one such disease that can affect one's nervous system. With this affecting their motor skills, those with advanced MS and use wheelchairs may have difficulty reacting quick to dangerous cliffs or drop offs when the chair approaches, resulting in wheelchair roll-over.

The University of Kansas Medical Center has sought out a solution for its members of the Multiple Sclerosis Achievement Center (MSAC). The University of Kansas Bioengineering program, the MSAC and the KU Mechanical Engineering department partnered to sponsor the Power Wheelchair Roll-Over Prevention System (ROPS), with a main objective to prevent wheelchair rollover when approaching floor drop offs such as curbs.

Goals

The original goal of this project was to improve the existing 2019-2020 Power Wheelchair ROPS system by adding a stopping mechanism, new LED indication system, and improve the buzzer function. We created these goals as it was originally assumed that the system would be able to be plugged in and have all components working. However, this was not the case and therefore, our overall goals changed.

Since the entire system did not work from the previous team, our team started from scratch. This included troubleshooting all six existing sensors, figuring out the correct wiring of all components, and buying new items to enhance the system. Because of all this change, the overall goal for the project was to get the system running in conjunction with all its components. In order to achieve this, we needed to successfully utilize eight LiDAR sensors and integrate them correctly to work with the existing LED indicators. These improvements will make the overall goal of creating a product that is suitable for commercialization use.

Criteria for Success/Failure

With our new goals, the first criterion for success will be if we can integrate two more LiDAR sensors into the system. It was decided that eight sensors would be necessary for full 360° coverage of the wheelchair. Two new sensors will need to operate in conjunction with the six existing sensors.

The next criterion will be if all components, existing and new, can create a unified system that works correctly on the wheelchair. The major component that needs to work correctly would be the LEDs. These are the best form of indicating potential danger to the user. Failure in this project would be caused by not completing either of these criteria for success.

Components/Instruments Needed

There are several different components that make up the device. The first component is the Raspberry Pi B+, a single board computer that receives (RxD) and transmits (TxD) data. This device is 67x56x11.5mm and contains an HDMI port, 3.5 mm audio-video jack, one USB port, camera serial interface, and display serial interface. Sensors and indicators can be connected to the Raspberry Pi using the 40 General Purpose Input/Output (GPIO) pins.

Figure 1: Raspberry Pi B+

The next component needed is the Big7 USB port expander in order to accommodate the extra GPIO pins when connecting all the sensors. The Big7 allows for 7 USB connections into one board. With 7 out of the 8 of the USB converters connected to the Big7 and the Big7 connected to the Raspberry Pi (the additional eighth sensor is connected directly to the Raspberry Pi), we were able to connect all the sensors to the Raspberry Pi.

Figure 2: Big7 USB Hub

As stated above, the use of a USB converter was needed. All eight LiDAR sensors were connected to the expander which communicates using transistor-transistor logic (TTL) through USB converters. The USB converter used was the 6 in One USB to UART Module to properly communicate the sensor data to the Big7 and then to the Raspberry Pi. This USB converter device contains 5V, Ground, TxD, and RxD connections. Each converter contains a top and bottom bridge, but the completed project only uses the top bridge.

Figure 3: USB to Serial Converter

One of the most important components of this system are the LiDAR sensors. The sensors used are TFMini Light Detection and Ranging (LiDAR) sensors. These sensors use laser pulses and a receiver to measure the distance at which objects are in relation to the transmitter on the sensor. Each TFMini LiDAR sensor has been strategically placed on the wheelchair with its detection constraints in mind. All TFMini sensors have a range of .3- 12 meters. But after testing it was found that the device will be reading data at a minimum of .5 meters. These sensors have a read accuracy of ±3 centimeters. All sensors will be plugged into their respective USB to Serial converters.

Figure 4: TFMini LiDAR Sensors

Another major component of this system is the LED indicator. The indicator used is a Pi Traffic Light. This indicator uses three 10 millimeter LED lights that are green, yellow, and red to give feedback to the user. When a distance between .43-.5 meters is read, the indicator will show

green as this is a safe reading. Depending on the distances read from the LiDAR sensors the coordinating LED light will light up, communicating with the user. This info can be seen below.

Figure 5: Pi Traffic Light LED Indicator

LED Color	Readings (meters)	Command
Green	0<d<=.50 [m] 0<d<=19.7 [in]	Safe
Yellow	.50<d<.66 [m] 19.7<d<25.9 [in]	Proceed with Caution
Red	d>=.66 [m] d>=25.9 [in]	Safety Hazard

Figure 6: LED Readings and Corresponding Commands

In addition to the LED indicator, there is also a buzzer that can be incorporated into the system. This device is the Active Piezo Buzzer Module. To connect the buzzer, 3 pins are used: voltage, ground, and a pin location to register sound. Our team was not able to fully incorporate the buzzer, however, it will be a necessary component for the overall project goal to be a success. Recommendations for buzzer improvement are included later in this report.

Figure 7: Active Piezo Buzzer

For this entire system to work, a battery that produces the voltage required by all of the components is necessary. The battery used for this system was the Anker PowerCore Portable Charger Battery. This battery has 3 USB ports which is enough to connect both the Big7 and Raspberry Pi to it, separately. The battery has a capacity of 26800 mAh which is enough to power all components. Duration of battery life was not able to be tested with our machine. Additionally, this battery is easy to charge as you just simply plug it into a wall outlet with its provided cord and it fully charges overnight.

Figure 8: Anker PowerCore Portable Charger Battery

Code

A python code is used in this project. The most up-to-date file, which is copied below, is called "8working2021.py". It is saved to the SD card plugged into the Raspberry Pi B+. Comments are inserted in red into the code below with explanations.

Import different programs and parameters for the code

```
from __future__ import division, print_function
import time
from tfmini import TFmini
from gpiozero import LED
```

Establish GPIO pins of the LED lights

```
red = LED(27)
yellow = LED(22)
green = LED(17)
```

Enter into main "while" loop

```
#print("haha")
while True:
```

Call first sensor (first sensor is called tf0)

```
# create the sensor and give it a port and (optional) operating mode
   tf0 = TFmini('/dev/ttyUSB0', mode=TFmini.STD_MODE)
   print('Device 0 Connected')
   try:
      #print("int try block")
      while True:
         d = tf0.read()
         print('Device 0 still Connected {:5}'.format(d))
         if d:
            print('Distance 0: {:5}'.format(d))
```

Print distance and react with parameters preset

```
            if 0<d<=.50:
               green.on()
               yellow.off()
               red.off()
            elif .50 < d < .66:
               green.off()
               yellow.on()
```

```
            red.off()
        elif d >= .66:
            green.off()
            yellow.off()
            red.on()


    #else:
        #print('No valid response')


        time.sleep(0.01)
        tf0.close()
        print('Device 0 closed. \n')
```
Close first "try" loop and move on to second sensor reading

```
  except:
    #print('through one')
    #tf.close()
```
Call second sensor and do the same thing as first sensor run

```
    tf1 = TFmini('/dev/ttyUSB1', mode=TFmini.STD_MODE)
    print('Device 1 Connected')


   #Commented out bc we copied to next section *dont uncomment unless you need to*
#   try:
#     #print("int try block 2")
#     while True:
#       d = tf.read()
#       print('Device 1 still Connected {:5}'.format(d))
#       if d:
#         print('Distance 1: {:5}'.format(d))
#

#         if 0 < d <= .50:
#           green.on()
#           yellow.off()
#           red.off()
#         elif .50 < d < .66:
#           green.off()
#           yellow.on()
#           red.off()
#         elif d >= .66:
#           green.off()
#           yellow.off()
#           red.on()
#

#     #else:
#       #print('No valid response')
#
```

```
#           time.sleep(0.1)
#           tf.close()
#           print('Device 1 closed. \n')
#

#    except:
#       #print('through 2')
#       #tf.close()
#

#       tf = TFmini('/dev/ttyUSB1', mode=TFmini.STD_MODE)


   try:
      #print("int try block")
      while True:
         d = tf1.read()
         print('Device 1 still Connected {:5}'.format(d))
         if d:
            print('Distance 1: {:5}'.format(d))


            if 0 < d <= .50:
               green.on()
               yellow.off()
               red.off()
            elif .5 < d < .66:
               green.off()
               yellow.on()
               red.off()
            elif d >= .66:
               green.off()
               yellow.off()
               red.on()


      #else:
         #print('No valid response')


         time.sleep(0.01)
         tf1.close()
         print('Device 1 closed. \n')


   except:
      #print('through one')
      #tf.close()
Call third sensor and do same thing
      tf2 = TFmini('/dev/ttyUSB2', mode=TFmini.STD_MODE)
      print('Device 2 connected.')
```

```
    try:
        #print("int try block")
        while True:
            d = tf2.read()
            print('Device 2 still Connected {:5}'.format(d))
            if d:
                print('Distance 2: {:5}'.format(d))


                if 0 < d <= .5:
                    green.on()
                    yellow.off()
                    red.off()
                elif .5 < d < .66:
                    green.off()
                    yellow.on()
                    red.off()
                elif d >= .66:
                    green.off()
                    yellow.off()
                    red.on()


        #else:
            #print('No valid response')


            time.sleep(0.01)
            tf2.close()
            print('Device 2 closed. \n')


    except:
        #print('through one')
#       tf.close()
```

Call fourth sensor and do same thing

```
        tf3 = TFmini('/dev/ttyUSB3', mode=TFmini.STD_MODE)
        print('Device 3 connected.')


    try:
        #print("int try block")
        while True:
            d = tf3.read()
            print('Device 3 still Connected {:5}'.format(d))
            if d:
                print('Distance 3: {:5}'.format(d))


                if 0 < d <= .5:
```

```
            green.on()
            yellow.off()
            red.off()
        elif .5 < d < .66:
            green.off()
            yellow.on()
            red.off()
        elif d >= .66:
            green.off()
            yellow.off()
            red.on()


    #else:
        #print('No valid response')


        time.sleep(0.01)
        tf3.close()
        print('Device 3 closed. \n')


  except:
    #print('through one')
#       tf.close()
Call fifth sensor and do same thing
    tf4 = TFmini('/dev/ttyUSB4', mode=TFmini.STD_MODE)
    print('Device 4 connected.')

  try:
    #print("int try block")
    while True:
      d = tf4.read()
      print('Device 4 still Connected {:5}'.format(d))
      if d:
        print('Distance 4: {:5}'.format(d))


        if 0 < d <= .5:
            green.on()
            yellow.off()
            red.off()
        elif .5 < d < .66:
            green.off()
            yellow.on()
            red.off()
        elif d >= .66:
            green.off()
            yellow.off()
            red.on()
```

```
        #else:
            #print('No valid response')


            time.sleep(0.01)
            tf4.close()
            print('Device 4 closed. \n')
    except:
        #print('through one')
#       tf.close()
```
Call sixth sensor and do same thing
```
        tf5 = TFmini('/dev/ttyUSB5', mode=TFmini.STD_MODE)
        print('Device 5 connected.')


    try:
        #print("int try block")
        while True:
            d = tf5.read()
            print('Device 5 still Connected {:5}'.format(d))
            if d:
                print('Distance 5: {:5}'.format(d))


                if 0 < d <= .5:
                    green.on()
                    yellow.off()
                    red.off()
                elif .5 < d < .66:
                    green.off()
                    yellow.on()
                    red.off()
                elif d >= .66:
                    green.off()
                    yellow.off()
                    red.on()


        #else:
            #print('No valid response')


            time.sleep(0.01)
            tf5.close()
            print('Device 5 closed. \n')

    except:
        #print('through one')
#       tf.close()
```

Call seventh sensor and do same thing

```
    tf6 = TFmini('/dev/ttyUSB6', mode=TFmini.STD_MODE)
    print('Device 6 connected.')


  try:
    #print("int try block")
    while True:
      d = tf6.read()
      print('Device 6 still Connected {:5}'.format(d))
      if d:
        print('Distance 6: {:5}'.format(d))


        if 0 < d <= .5:
          green.on()
          yellow.off()
          red.off()
        elif .5 < d < .66:
          green.off()
          yellow.on()
          red.off()
        elif d >= .66:
          green.off()
          yellow.off()
          red.on()


    #else:
      #print('No valid response')


      time.sleep(0.01)
      tf6.close()
      print('Device 6 closed. \n')




  except:
    #print('through one')
#     tf.close()
```

Call eigth sensor and do same thing

```
    tf7 = TFmini('/dev/ttyUSB7', mode=TFmini.STD_MODE)
    print('Device 7 connected.')


  try:
    #print("int try block")
    while True:
```

```
            d = tf7.read()
            print('Device 7 still Connected {:5}'.format(d))
            if d:
               print('Distance 7: {:5}'.format(d))


               if 0 < d <= .5:
                  green.on()
                  yellow.off()
                  red.off()
               elif .5 < d < .66:
                  green.off()
                  yellow.on()
                  red.off()
               elif d >= .66:
                  green.off()
                  yellow.off()
                  red.on()


         #else:
            #print('No valid response')


            time.sleep(0.01)
            tf7.close()
            print('Device 7 closed. \n')



      except:
         #print('through one')
         tf7.close()
#except:
   #print('close')
```

Code ends with 8 distances printed and restarts over to sensor 1

Experimental Setup & Connections

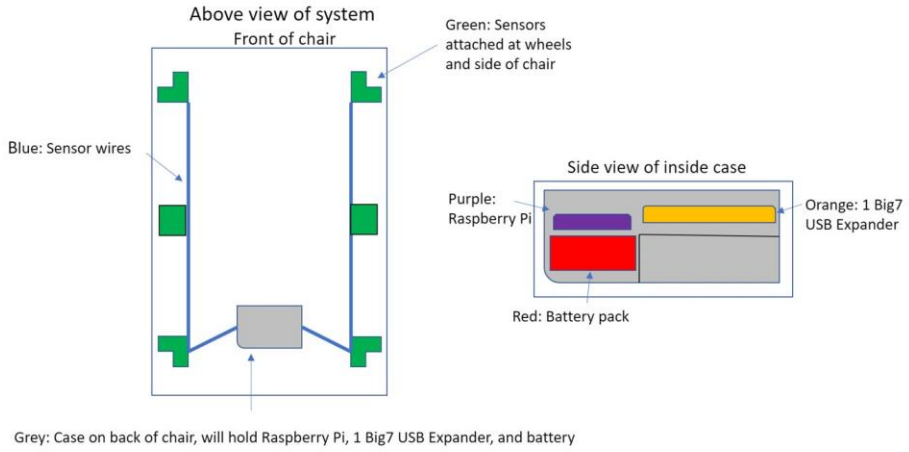


Figure 9: Basic Diagram of System Around Chair and in Case

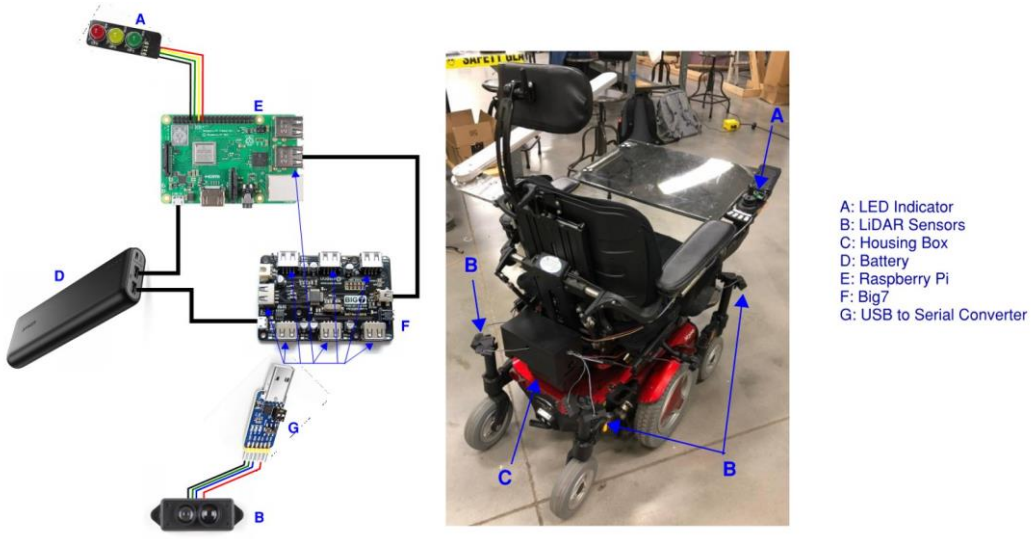


Figure 10: Wire Diagram and Outer Components on Wheelchair

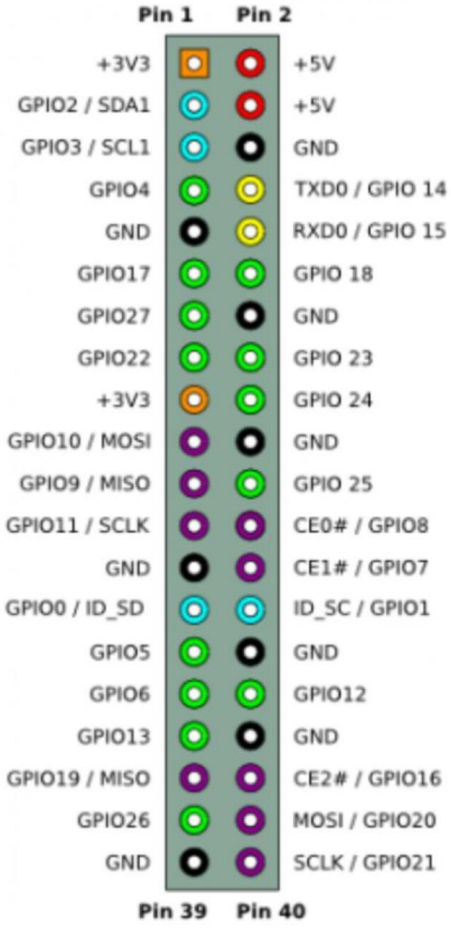


Figure 11: GPIO pinout diagram for Raspberry Pi

Step-by-Step Procedure

1. Make all the proper connections using the connection setup diagram.
    a. Make sure the Big7 and the Raspberry Pi B+ have adequate power by properly connecting the Anker PowerCore Battery.
    b. Be sure to use a monitor, keyboard, and mouse to properly access the Pi software and read the data value readings being displayed.
    c. Connect the Raspberry Pi to the power source last.
    d. Be sure the Raspberry Pi sits securely inside its case. If the underside pins are exposed, it could risk shorting the circuit.
2. After the connections are made, the monitor should show the Pi booting up. It will pull up a software called Thonny where all the python (.py) files are stored on the SD card.
3. The most up-to-date version of the code is the 8working2021.py. This can be opened and edited by going to the file manager in the top left corner, then going to Documents and clicking on the Code subfolder within the Capstone folder.

4. To run this code, go to the top left corner and open the terminal. In the terminal window, enter the commands: “cd Documents”, “cd Capstone”, “cd Code”, “python 8working2021.py”.
    a. If the components connections are correct, the terminal will display readings as Distance0, Distance1, Distance2, Distance3, Distance4, Distance5, Distance6, Distance7. It will also display the sensors labeled as USB0 to USB7 depending on the order which the converters are plugged into the Big7 USB expander.
    b. If the terminal shows an error message, one can verify the Pi is reading all the required devices by running the “ls usb” command in the terminal.
5. After verifying the code is correctly running manually, the next step is to run the code automatically on the Pi as it starts up. Ultimately, the Pi is connected to the monitor and commands are entered into the terminal that tell it the run the script automatically after the Pi reboots. When this works, the Pi should be able to be unplugged from the monitor and into the box attached to the back of the chair to run automatically. There are different methods to do this, the init.d method is likely the best one for this project. *NOTE: The autorun ability on this project still needs further troubleshooting to correct inconsistencies with boot sequence initialization before the script runs. The following instructions explain the init.d method, and with this, further troubleshooting will be needed. *
    a. Using the terminal, add the 8working2021.py file to the init.d directory using the following line
        i. “sudo cp /home/pi/8working2021.py /etc/init.d/”
    b. Move to the init directory and open the script:
        i. Type command “cd /etc/init.d”, *press enter*, type command “sudo nano 8working2021.py” *press enter*.
    c. Enter the following lines to the top of the 8working2021.py script in the init directory:

```
### BEGIN INIT INFO
# Provides: 8working2021.py
# Required-Start: $all
# Required-Stop: $all
# Short-Description: run 8working2021.py on boot
### END INIT INFO
```

    d. Make the script in the init directory executable by changing its permission
        i. “sudo chmod +x 8working2021.py”
    e. Run the command:
        i. “sudo update-rc.d 8working2021.py defaults”

f. Reboot the Pi to see if the script runs automatically
    i. “sudo reboot”
6. When finished using the Pi and turning off the Raspberry Pi, be sure to do so by running the command “sudo shutdown now” in the terminal.
    a. Do not turn off the Pi by unplugging the power source from it. It could possibly break it.

Experimental Testing & Progress

The first thing that was tested was the code used to run the system. However, while testing the code, we ran into several issues with the wiring of the sensors and the code itself. After changing the wiring of one sensor and running the code again, it was found that the only sensor providing data was the one that had been changed. All sensors were rewired in the same way and began providing data individually. To figure out why the sensors would not all work at the same time, we contacted Thomas Woodruff, an Electrical Engineering Graduate from KU. He was able to help us fix the code and troubleshoot each of the 6 sensors individually so that they could successfully read distances. This allowed us to order the 2 additional sensors and set them up properly.

With every sensor reading distances, we used a piece of wood held at various predetermined heights to measure the minimum distances of each sensor and found that the sensor placements on the chair would need to be adjusted. Designs were then created to raise the sensors enough to allow the ground to be just beyond the minimum reading distances of each. This change will allow the sensors to provide more accurate distance readings to properly notify the user of the chair of any drop-offs.

Once the sensors were set to their proper heights and the distances were updated in the code and we began testing the lights. With all 8 sensors running, objects were held in front of each at varying distances. As the distance of the objects changed, each sensor was able to detect the change and illuminate the green, yellow, and red light corresponding to the distance.

Future Recommendations for Improvement

The most important way to improve this system is to wire in 360° RGB LED lights around the wheelchair control panel. The current light configuration notifies the user of how close the drop-off is to the chair but does not show the direction in which the drop-off was detected. These lights will provide the user a direct range of vision and more control over the chair by showing the exact position of any drop-off that is detected by the sensors.

Another important issue that should be investigated is the current buzzer associated with this system. We ran into problems using the buzzer with the system because when it is plugged in, it will immediately start up and continue buzzing. After spending time attempting to get it to work properly, we decided that the easiest solution for this is to get a new buzzer. The current buzzer is under-sized and likely not loud enough to successfully alert the user of a drop-

off in real world conditions, so purchasing a new one seems to be the best solution. It also should be considered to put the buzzer near the head of the chair so the user can hear it clearer.

Creating fixtures to raise each of the sensors is something that should also be worked on as well. We have created PVC prototypes to raise the sensors to the proper height so the sensors will provide more accurate readings, but we recommend 3D printing the final version of these fixtures to the same dimension as the prototypes that are currently on the chair. 3D printing these fixtures will help them be more stable and allow them to be directly mounted to the chair.

Knowing the length that the battery will run would be something else that would need to be tested. Running all the sensors with a fully charge battery will be useful for the end user so they know how often to charge it. If possible, the next group should consider buzzer and LED battery usage.

The thing we were unable to get to this year was waterproofing the wires. The wires coming from the rear box to the sensors will need to be waterproofed to withstand real world situations. The best suggestion for this is to find waterproof hollow tubing (such as rubber) and feed each of the wires through it. Another option is to run the wires through waterproof heat shrink tubing.

Conclusion

In summary, we believe that the overall goals for this project were met. Our team was able to enhance the existing Wheelchair Roll-Off Prevention from the 2019-2020 school year. The enhancements included the successful utilization of eight LiDAR sensors for a full range of coverage and ability to use the LED indicator lights in conjunctions with the sensors. Completing these two goals allows for an increased user friendliness and for the system to work better as a whole.

Our team believes that we have left an even better foundation for the next group that works on this project. With our organized documentation, detailed pictures, running code, and updated components, the future of the Wheelchair Roll-Off Prevention system is optimistic. We believe that our team brought the whole project many steps closer to achieving the original goal of the project: creating a Roll-Off Prevention system for commercialization use so that patients with MS can feel safe in their wheelchair.