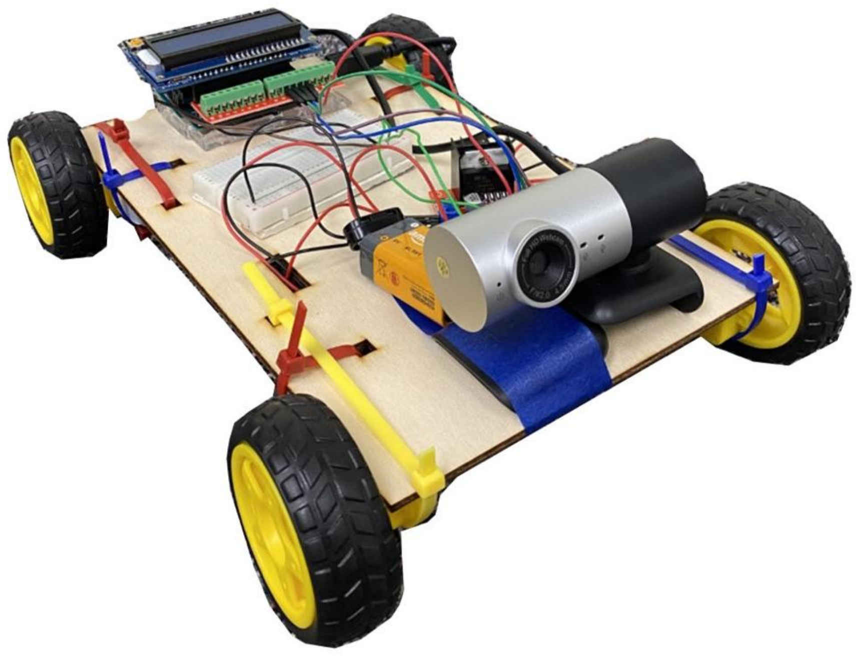
Using YOLO Object Detection to Control a Car with an Arduino



Preston Ito
Dr. Kuchera
December 9, 2022
Electronics and Instrumentation

Introduction and Motivation:

For my final Electronics project, I coded and built a car that moves depending on what it sees with an external USB camera that is attached to it. The car uses YOLO object detection to scan what it sees, and if it sees a single person, it will move toward them. The car also has an LCD display that will say what it's doing, whether that be turning left or right, or moving forward. If it's not moving (it won't move if it detects anything but a single person), the LCD will display what it sees and how many it sees. This project could be expanded to perform a wide range of robotic and autonomous functions, such as creating a robot vacuum to navigate a room, robot waiters in restaurants, or potentially even autonomous robots that can explore areas that could be hard to get to for humans. It could also even be used for less practical purposes, such as a cat or dog toy that could follow around a child.

Background and Theory:

This project can be broken down into two categories: hardware and software. The hardware includes things such as building the frame and assembling the different components. The software side of the project includes the YOLO object detection code and the Arduino code. The basis of this project was inspired by a previous project, titled *Bluetooth Controlled Car*, on the Arduino Project Hub website.

Hardware

As for assembling this project, I used the following parts:

- Arduino UNO
- Adafruit 16x2 RGB LCD Shield
- Gikfun Screw Shield Expansion Board for Arduino UNO
- 9V battery
- 9V battery to wires adapter
- ShangHJ 4 Sets DC Gearbox Motor Kit (includes 4 wheels, 4 motors, and motor driver)
- External USB camera
- Mini breadboard
- Various wires (jumper and normal)
- Wood frame
- Zip Ties
- Tape
- Computer with 2 USB ports

The LCD screen was used to display what was being seen and what actions it takes as it's taking them. Since the LCD fits right on top of the Arduino, I used the Gikfun Shield Expansion to allow me to cleanly plug more wires in the Arduino. The 9V battery (with the 9V battery to wire adapter) was used in conjunction with the 4 wheels, motors, and the motor driver to power all of the motors. An external USB camera was used and sat on top of the car while plugged into my laptop. For the car's frame, I used a ⅛" thick wooden sheet that I cut to fit the specifications of the other components. I used Zip Ties and tape to hold the components with the car's frame.

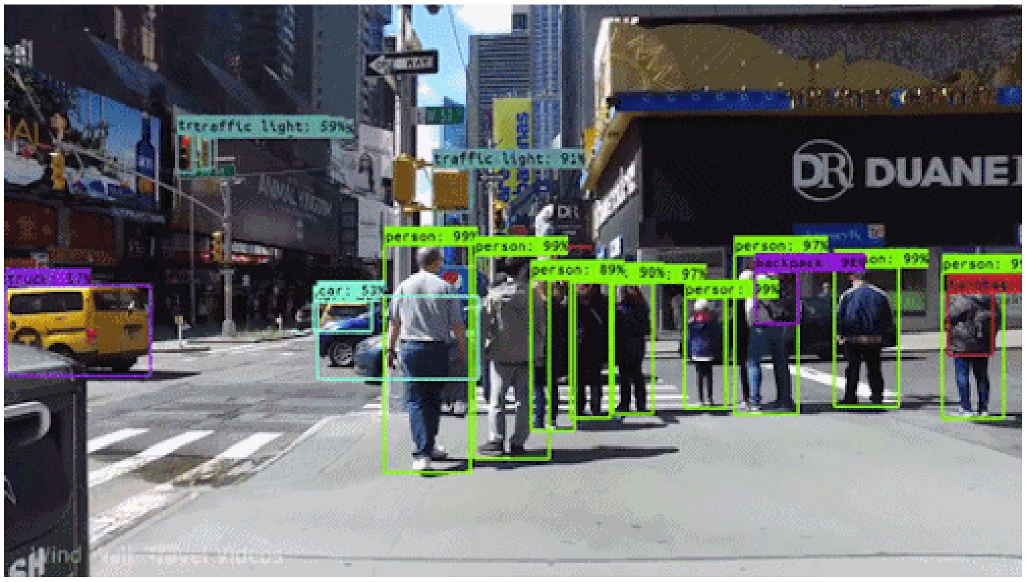
Software

On the software side of things, I had two main pieces of code:

- Arduino Code
- YOLOv5 Object Detection Code

The Arduino code was largely written by myself, but the fundamentals were inspired by the *Bluetooth Controlled Car* project. Essentially, I had to write an Arduino file that could receive and understand information that was sent from the completely separate Python YOLOv5 program, and write different actions depending on what was being read. Some of these actions include (but aren't limited to) turning left or right, or writing different messages to the LCD display. The Arduino code also required some external package installation, including the Adafruit package which allowed me to write to the LCD display, and the PySerial package which allowed for communication between the Arduino file and the Python YOLOv5 file. To actually use the PySerial package, I had to `import serial` at the top of the YOLOv5 file and also create a `write_read()` function that I used from the Arduino Project Hub website in a project titled, *Serial Communication between Python and Arduino*.

The YOLOv5 Object Detection Code was a package that was not created by me, and was created by Ultralytics. YOLO, which stands for "You Only Look Once," was developed in 2015 by Redmon, Divvala, Girshick, and Farhadi. YOLO is currently one of the fastest methods of object detection, making it ideal for usage in real-time applications, such as self-driving cars, robotics, medical imaging, and surveillance systems. It uses a neural network that's designed to process an image just once and detect several images with them simultaneously. A neural network is a machine learning algorithm that is able to analyze deep, complex features of data and output accurate predictions. YOLOv5's machine learning algorithm was trained with the COCO (stands for Common Objects in Context) dataset, which is essentially a large-scale dataset often used for basic object detection algorithms. Training a neural network in this case basically means feeding it an extremely large number of images with accurate detections and bounding boxes so that it can successfully reproduce the accurate results given any new image. Ultimately, YOLO can take any image, video file, or video stream and detect objects, and create a bounding box around each object with the confidence of its detection in the form of a float between 0 to 1. Here is an example animations of what YOLO looks like:



This open-source package along with instructions on how to download it with pip (pip is a Python package manager) are all available on Github. Although it wasn't written by me, I still had to learn and be somewhat familiar with the functionality of the Python file in order to edit where necessary and send correct messages to the Arduino file.

Procedure:

My final project changed quite a bit from my initial project proposal. This was due to unforeseen obstacles that made it nearly, if not impossible to build my project as stated in the proposal. I approached this project in phases, starting with getting familiar with the capabilities and limitations of the software and hardware I planned to use, adapting and piecing together what I could, and final testing and improvements.

Phase 1: Testing basic functionality (software)

The first thing that I did when starting the project was focus heavily on the software side of things. I wasn't completely sure that I was going to be able to create the project as stated in my proposal, which was to "build a bluetooth remote controlled car that has a camera that uses YOLO for object detection." I could get YOLOv5 to work on my own computer, but had idea how I would get it to interact with the Arduino. Some of the most basic software first steps I took included the following:

- ☑ Make sure that YOLO works with the external USB camera
- ☑ Figure out how to execute the YOLO file in Visual Studio Code
- ☑ Find where in the code it outputs the objects detected and figure out how to print these detections to the terminal
- ☑ Figure out how to communicate between Python file and Arduino

To run the YOLO file in VS Code, I found the `detect.py` file and ran it with the “Run Python File " button on the top right in VS Code. Once I ran it once, the line `/usr/bin/python3 /Users/prestonito/yolov5/detect.py` appeared in the terminal, and I was able to rerun the file by simply clicking on the up-arrow and hitting enter. To use an external USB camera as the video stream source, I just had to add the argument `-source 1` to the end of the line in the terminal. Finding where the code actually outputs the object was a little more difficult and took some trial and error. I eventually found this in lines 175-178 in the file, as shown below:

```
175            # Print results
176            for c in det[:, 5].unique():
177                n = (det[:, 5] == c).sum()  # detections per class
178                s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string
```

Essentially, line 178 puts everything together in a single string, including how many of each object and what the object actually is. The for-loop as indicated on line 176 goes through and does this for each detection per image. I created an object that was just the detections and how many of each detections and called it `actualObjects` of type String, shown in line 182. I then printed this object to the terminal just to keep track of everything it sees.

```
180                # Creates an object that is a string that is the string of the
181                    number of how many things it sees and what it sees
182                    actualObjects = f"{n} {names[int(c)]}{'s' * (n > 1)}"
183
184                # Prints the string in the terminal so I can see what it thinks
185                    it's seeing
186                    print("actualObject: " + actualObjects)
```

Now that I had found how to print exactly what I wanted to send over to the Arduino, I had to figure out how. This was one of the biggest obstacles I faced as I had no idea how to achieve this, if even possible. I first tried to use VS Code instead of the Arduino IDE. I had hoped that communicating between the two files would somehow be easier if they were in (roughly) the same environment. I ended up successfully doing this with an extension called *PlatformIO* on VS Code. However, I still ran into the same problem with no real way to communicate between files. The next thing I tried was to look into getting the Python code to write to a .txt file that the Arduino code could read. Unfortunately, it turned out that Arduino’s can’t read .txt files directly from the PC. Luckily, I received a suggestion from Dr. Michelle Kuchera to look into a specific package that could allow for communication between the two files. This is where I found the Pyserial package, a package that would allow for Python code to write to Serial, which the Arduino code can read. With this new import in the Python program, I could write a function that sends a string to the Arduino through Serial. I got this function from the example on the Arduino Project Hub website. I also had to include the specific port and baud rate in the file in order to give it a destination when writing.

```
        arduino = serial.Serial(port='/dev/cu.usbmodem14201',
        baudrate=9600, timeout=.1)
    # Preston's edits: A function that writes the information to the Arduino
        def write_read(x):
            arduino.write(bytes(x, 'utf-8'))
            time.sleep(0.1)
            data = arduino.readline()
            return data
```

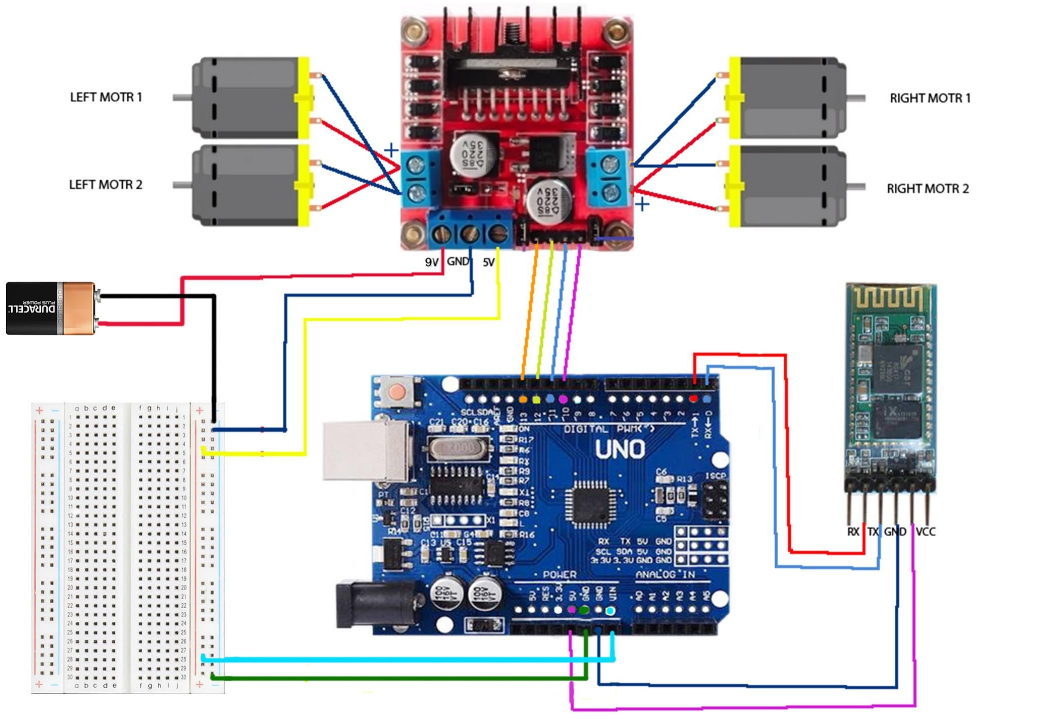
The function takes in a string and uses the `.write` function to send the string to the Arduino. The Arduino can then simply read the string with a `detection = Serial.readString();`.

Phase 2: Testing basic functionality (hardware)

On the hardware side, I also had to make sure the physical components worked as desired. While I'd worked with an Arduino UNO before, there were many components I was unfamiliar with prior to starting this project. Some steps I took in order to test and get comfortable with the different components were:

- ☑ Follow the circuit schematic of the *Bluetooth RC Car* and test the motors
- ☐ Connect the HC-05 Bluetooth module to my computer or phone
- ☑ Successfully display letters and numbers on the LCD display.

I slightly modified the circuit schematic from the example project to remove the headlight LED and include a mini breadboard. My altered circuit diagram consisting of the motor driver and motors, 9V battery, mini breadboard, Arduino UNO, and the HC-05 bluetooth module is shown below.



This is where I ran into my first big issue. As done in the example project, I tried to download an app that would allow me to control the motors from my phone. However, after downloading 4 different apps, I found no success with connecting to my phone. The HC-05 bluetooth module was blinking red, which indicated that it was trying to connect, but turns out that this particular bluetooth module is incompatible with iOS devices. I tried to connect the bluetooth module with my Macbook Air but also found no success. Furthermore, I realized that I couldn't connect the bluetooth module while also connecting the Arduino to my laptop with a USB cable. The bluetooth module takes up a port, and on the Arduino IDE, only one port can be selected at a time. This makes some intuitive sense, as one can imagine the complications that could occur when trying to send data to a single Arduino through two different ports, especially if at the same time. Basically, I couldn't keep the bluetooth module wired to the Arduino and go about testing my code at the same time. Therefore, I completely removed the bluetooth module for the time being and decided to investigate it further if I had extra time before the end of the semester.

Besides the bluetooth module, I also tested the motor functionality with a simple Arduino script:

```
void setup() {
 // connects the pins to the corresponding motor functions
 pinMode(13, OUTPUT);    //right motors forward
 pinMode(12, OUTPUT);    //right motors reverse
 pinMode(11, OUTPUT);    //left motors reverse
 pinMode(10, OUTPUT);    //left motors forward
}
```

```
void loop() {
 // turn all motors on
 digitalWrite(13, HIGH);
 digitalWrite(10, HIGH);
}
```

The `setup()` function starts by assigning the pin numbers to the motors as output pins. Then, the `loop()` function keeps turning the motors on, writing HIGH to the right and left motors forward. At first, the motors weren't moving at the same speed, which wouldn't have allowed the vehicle to move straight. However, it turned out that that was just due to a weak 9V battery.

I also had to test the LCD display. This display was actually an addition to my project from the initial proposal, because I figured that it shouldn't be too hard to use and also because it would be nice to include some form of communication between the Arduino and the user (other than the terminal output on my laptop). As mentioned before, I had to download the Adafruit LCD libraries and include them in the header of the Arduino file. I also followed a basic tutorial and included some simple lines to test the display. Now, my testing file looks like this:

```
#include <Adafruit_RGBLCDShield.h>
#include <utility/Adafruit_MCP23017.h>
void setup() {
 // set up the LCD's number of columns and rows
 lcd.begin(16, 2);
 lcd.clear();
 lcd.setBacklight(WHITE);
 // connects the pins to the corresponding motor functions
 pinMode(13, OUTPUT);    //right motors forward
 pinMode(12, OUTPUT);    //right motors reverse
 pinMode(11, OUTPUT);    //left motors reverse
 pinMode(10, OUTPUT);    //left motors forward
}
void loop() {
// print "hello world" on the LCD display
lcd.print("hello world");
 // turn all motors on
 digitalWrite(13, HIGH);
 digitalWrite(10, HIGH);
}
```

When written to the Arduino, the LCD display successfully displayed “hello world”. Some LCD displays seemed to work better than others, so I tested different ones and used the one I found the most consistent.

Phase 3: Piecing hardware and software together

With the initial testing phase done, the next phase focused on piecing these different components together. Especially with an incompatible bluetooth setup, I had to alter my original project proposal. The next main goals of this phase were to:

- ☑ Send messages from YOLO file to LCD display
- ☑ Think of a way to control car’s movement without bluetooth
- ☑ Laser Cut a frame for the car

To send messages from the YOLO file to the Arduino to send to the LCD display, I used the `write_read(x)` function and the `actualObjects` object that were referenced earlier in *Phase 1*. Everytime the YOLO file reached the point in the code where it created a new `actualObjects` object, I used the `write_read(x)` function with the `actualObjects` as the parameter. The code looks like this:

```
actualObjects = f"{n} {names[int(c)]}"
write_read(actualObjects)
```

To check if this was successfully sending over to the Arduino, I included a some lines in the Arduino code (in the loop, not the setup) that prints the incoming message to the LCD display:

```
if (Serial.available()) {
   detection = Serial.readString();
     lcd.clear();
     lcd.setCursor(0, 0);
     lcd.print(detection);
}
```

These 2 additions to the code somewhat worked. There seemed to be some timing issues in the communication between the two files. The LCD display would display multiple strings at the same time. For example, if the YOLO made 2 detections in two frames in a single second, such as “1 person” and “1 person, 1 cup”, ideally, the LCD display would first show “1 person”, clear the display, and then show “1 person, 1 cup”. However, the LCD acted quite inconsistently, sometimes showing “1 person1person, 1 cup”, or even displaying something inexplicable such as, “1person1person1person1person1cup”. Although this behavior was not what I wanted for the final product, I decided to move on since I thought I had more pressing issues to focus on before I fine tuned it.

One of the more pressing issues was figuring out what to do to control the car. My next thought was to create a “remote” using 4 pushbuttons (one to control going forward, one to control going left, one to control going back, and one to control going right). However, I would’ve had to walk alongside the car with the wires from the buttons connected to the Arduino. While I think this would’ve been possible, I wanted to consider other options. Another option I considered was controlling the car with my laptop through the Serial monitor. For example, by using the WASD keys on my keyboard, I could control the car in four directions. Unfortunately, in trying to code this, I realized that this would not be possible. For some reason, I couldn’t open the Serial monitor while the YOLO code was running. It would give me an error that said “*Couldn’t open Serial monitor, port is busy*”. It appeared that since the YOLO file was connected to the Serial monitor of that specific port, I couldn’t open the Serial monitor anywhere else. So, I considered using two Arduinos that would be connected through two different ports. However, my laptop only had two USB ports, and one of them had to always be connected to the USB camera. I also considered using a USB hub that would allow for more connections, but I didn’t have immediate access to one to test. I ended up figuring that there is a good chance that it wouldn't work, even if I did have one readily available, because it would just be going through one port in the end. After all this, I decided to once again alter my original proposal. Instead of having manual control of the car as I originally intended, whether that be through some kind of remote control, I decided to control the car depending on what it saw through YOLO. Ultimately, I wanted to make the car move forward and turn left or right to follow a person, if it detected a person.

The first step toward this new goal was to turn the motors on based on a specific message received from the YOLO output, specifically a single person. The way I did this was with an if-statement in the Python code:

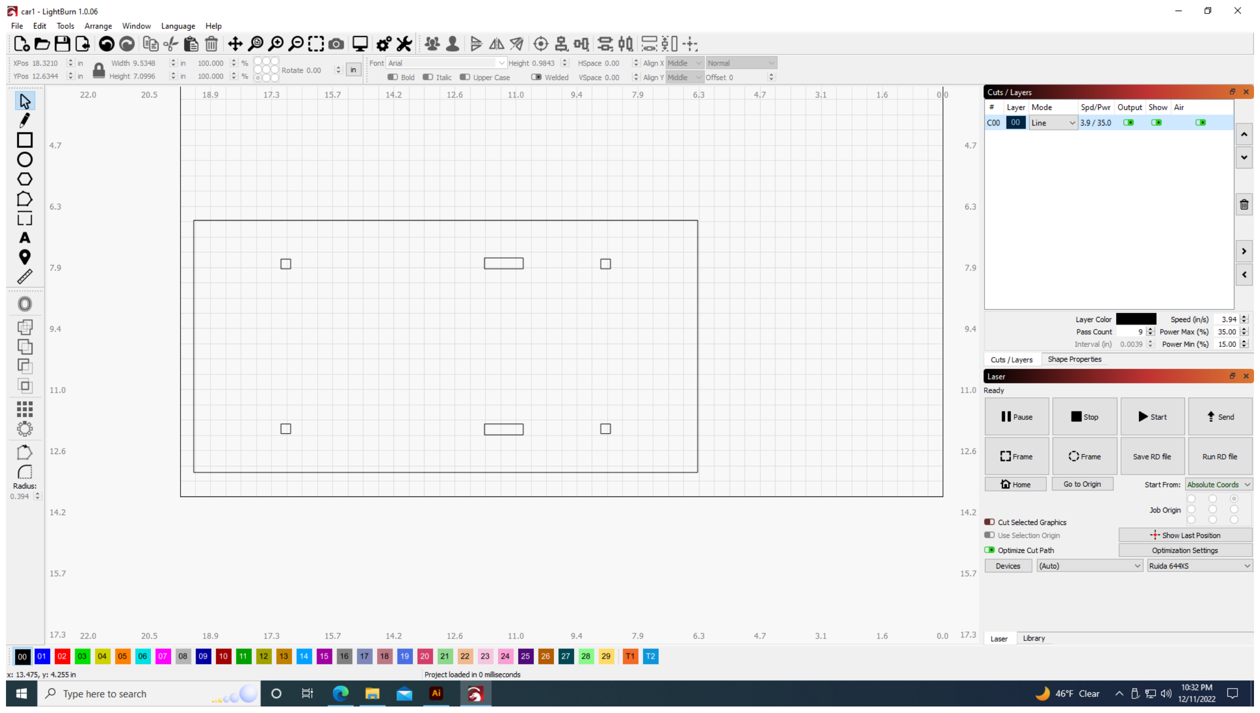
```
if actualObjects == "1 person":
    write_read(actualObjects)
```

Basically, if the output of the YOLO code was “1 person”, only then should it be sent to the Arduino file. Then, in the Arduino file, I included another if-statement just to add another filter:

```
detection = Serial.readString();
if (detection == "1 person") {
  digitalWrite(13, HIGH);
   digitalWrite(10, HIGH);
}
```

With these two modifications, the code worked; the motors turned on if YOLO detected 1 person. Otherwise, nothing would happen. While many more modifications would have to be made in order to make the car follow a person around, this would suffice for now and I moved on to the next task: creating a frame for the car.

Since I work in the Makerspace as a student employee, I was familiar with using the Laser Cutter. I planned to measure and design a simple frame, with holes to allow for Zip Ties to hold the motors to the frame and separate holes to allow for the wires from the motors to connect directly to the motor driver on top of the frame. On top of the frame, it would hold the Arduino, camera, motor driver, and 9V battery. Here is a screenshot from the Laser Cutting software of my first design:



XPos 18.3210 in, YPos 12.6344 in, Width 9.5348 in, Height 7.0996 in, 100.000 %, 100.000 %, Rotate 0.00
Font Arial, Height 0.9843, HSpace 0.00, VSpace 0.00, Align X Middle, Align Y Middle, Normal, Offset 0

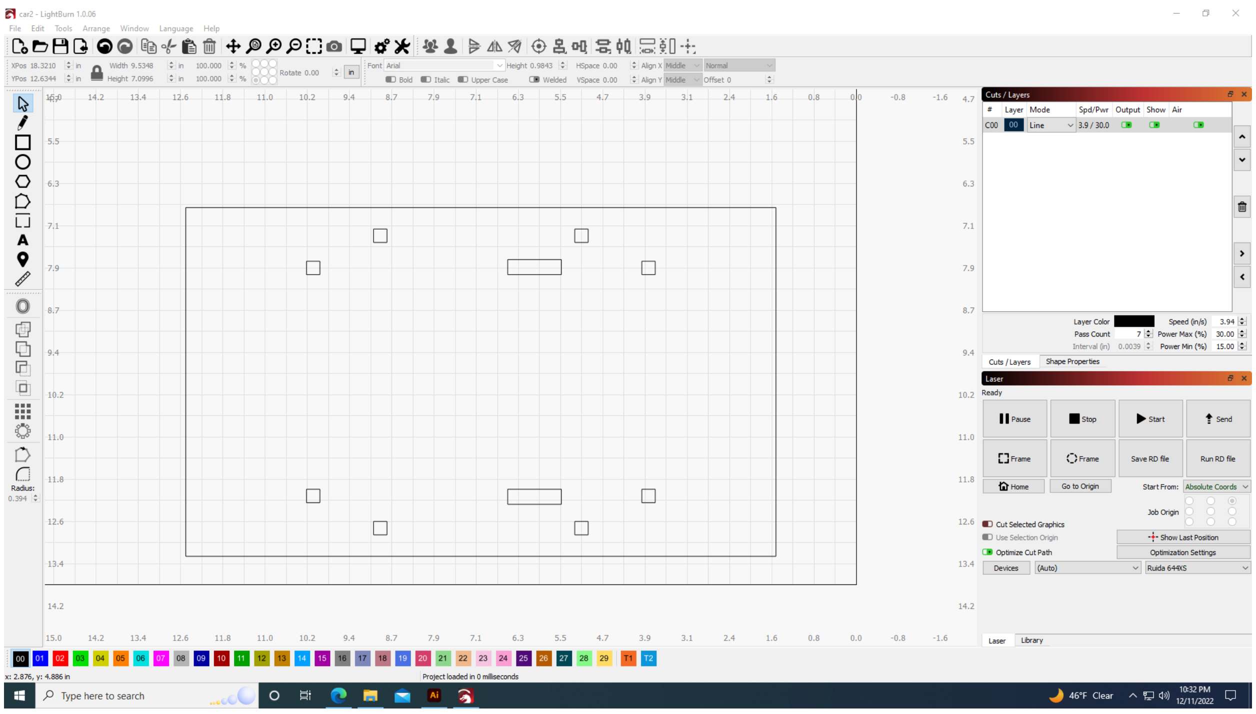
#	Layer	Mode	Spd/Pwr	Output	Show	Air
C00	00	Line	3.9 / 35.0	🟩	🟩	🟩

Layer Color ■, Speed (in/s) 3.94
Pass Count 9, Power Max (%) 35.00
Interval (in) 0.0039, Power Min (%) 15.00

Start From: Absolute Coords
Job Origin
Cut Selected Graphics
Use Selection Origin
Optimize Cut Path
Show Last Position
Optimization Settings
Devices (Auto) Ruida 644XS

x: 13.475, y: 4.255 in
Project loaded in 0 milliseconds
46°F Clear 10:32 PM 12/11/2022

There were a couple issues with my first frame that I cut. First, it was longer than it needed to be, which added extra weight and length that made it harder for the motors to move and turn sharply. Secondly, I noticed that a single Zip Tie wasn't enough to hold the motors in place that well. The next frame I designed was several inches shorter and also included another hole to add a second Zip Tie, perpendicular to the original Zip Tie, for each motor:


car2 - LightBurn 1.0.06

XPos 18.3210 in Width 9.5348 in 100.000 % Rotate 0.00
YPos 12.6344 in Height 7.0996 in 100.000 %

Font Arial Height 0.9843 HSpace 0.00 Align X Middle Normal
Bold Italic Upper Case Welded VSpace 0.00 Align Y Middle Offset 0

#	Layer	Mode	Spd/Pwr	Output	Show	Air
C00	00	Line	3.9 / 30.0	✓	✓	✓

Layer Color ■ Speed (in/s) 3.94
Pass Count 7 Power Max (%) 30.00
Interval (in) 0.0039 Power Min (%) 15.00

Laser: Ready
Start From: Absolute Coords
Cut Selected Graphics
Use Selection Origin
Optimize Cut Path
Devices (Auto) Ruida 644XS

x: 2.876, y: 4.886 in
Project loaded in 0 milliseconds

This second and final frame worked well–the motors were much more stable and the car was about as short as possible while still being able to fit everything on top of it.

Phase 4: Fine tuning and improvements

At this point, my car worked on a very basic level. Not only was it now one assembled piece with the addition of the frame, but the motors turned on, moving the car forward, if and only if the YOLO detected "1 person". That being said, I still had improvements to make:

- ☑ Make the car turn left or right depending on where it detects the person
- ☑ Fix the LCD display (make it display one thing at a time)
    - ☑ Make the display say what it's doing if it's turning or moving forward

The logic of turning the car would be simple: if the person was to the right, turn right. As shown in the example animation in the *Background and Theory* section, a bounding box is created for each object it detects. The challenge was figuring out how to access some information about the bounding box that it creates in the video stream output. After some trial and error, I found the coordinates of the bounding box in this block of code:

```
                for *xyxy, conf, cls in reversed(det):
                    if save_txt:  # Write to file
                        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) /
                                gn).view(-1).tolist()  # normalized xywh
```

Printing the `xywh` object showed that it was a list that contained 2 objects: the x and y-coordinates. I edited this block of code to just take the x-coordinate (for left/right control, I don't care about the y-coordinate) from the list, turn it into a string, and write it to the Arduino, as shown below.

```
                for *xyxy, conf, cls in reversed(det):
                  coordinates = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) /
                                 gn).view(-1).tolist()
                  x_coord = coordinates[0]
                  x_coordStr = str(x_coord)
                  write_read(x_coordStr)
```

Once I had this, the next step was to add code to the Arduino to read these messages and send signals to the motors accordingly. By running YOLO and monitoring the output coordinates of the bounding boxes, I could deduce that the coordinates ranged from 0 to 1, with the origin being the bottom left corner. Therefore, if the x-coordinate was less than or equal to 0.4, I sent the car to turn left. Likewise, if the coordinate was greater than or equal to 0.6, I sent the car to turn right. If the x-coordinate was in between 0.4 and 0.6, I moved the car forward. Mechanically speaking, the car turned by moving either the left or right wheels forward while the opposite wheels turned back. This caused the car to turn in place. For instance, to make the car turn left, the two wheels on the right side of the car moved forward while the two wheels on the left side simultaneously moved backward. One thing to note is that since all of the messages were being sent by the `write_read` function, the Arduino file had to filter through all of the messages to figure out exactly what message it was reading; specifically whether it was an object detection or an x-coordinate. I did this by using nested if-statements and doing several readings from Serial. Basically, if the first read from Serial is "1 person", then do another read from Serial. Now, if the reading is a coordinate (basically if it starts with the characters "0."), turn the string into a float, and use if statements to figure out how the car should move accordingly. If it doesn't read "1 person", then it will print whatever it sees instead of going into these nested if-statements. Note that this code also writes to the LCD display as it moves in different directions :

```
detection = Serial.readString();  // reading string from Serial, which in this
case, comes from the Python file

// moves/turns the car only if the only thing it's reading is "1 person"
if (detection == "1 person") {
  xCoordStr = Serial.readString();
     // only takes strings that are clearly coordinates (starts with 0.)
     if (xCoordStr.substring(0,2) == "0.") {
        xCoord = xCoordStr.substring(0, 5).toFloat();
       // turns the car left if the x-coordinate is to the left
       if (xCoord <= 0.4) {
         lcd.print("Turning left");
         digitalWrite(13, HIGH);
         digitalWrite(11, HIGH);
       }
       // turns the car right if the x-coordinate is to the right
       else if (xCoord >= 0.6) {
         lcd.print("Turning right");
         digitalWrite(12, HIGH);
         digitalWrite(10, HIGH);
       }

       // if the x-coordinate of the human is in the range of 0.4-0.6, move
forward
       else {
         lcd.print("Moving toward");
         lcd.setCursor(0, 1);
         lcd.print("person");
         lcd.setCursor(0, 0);
         digitalWrite(13, HIGH);
         digitalWrite(10, HIGH);
       }
     }
   }
```

This block of code is the majority of my final Arduino file. It is successfully able to read from the Python file, and if it detects "1 person", it will turn right or left or move forward to follow the person.

In regard to the LCD display issue of printing several strings at once, I realized that this was mostly a result of two things: the LCD screen printing the same object twice if consecutive detections are the same, and timing issues between the Arduino file and the Python file. To approach the first problem, I put all of the detections in a set. A set is a data structure that has no order and doesn't take duplicates. For example, if I add "1 cup" to the set and then try to add "1 cup" again, the set will still only contain one element: "1 cup". Then, print all of the elements in the set. This should help in taking care of the repeated print on the LCD screen. To solve the second issue, I included several `time.sleep(1.5)` lines in the Python file whenever it used the `write_read()` function. This basically just pauses the code for 1.5 seconds before moving on to the next line. A block of code that uses these two modifications are shown below:

```python
# Make a set to store all detections (to prevent printing the same thing twice)
betterObjects = set()
for c in det[:, 5].unique():
    n = (det[:, 5] == c).sum()  # detections per class
    s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "
    actualObjects = f"{n} {names[int(c)]}"
    write_read(actualObjects)
    print("actualObject: " + actualObjects)
    if actualObjects == "1 person":
        write_read(actualObjects)
        time.sleep(1.5)
    else:
        betterObjects.add(actualObjects)
        for word in betterObjects:
            if word != "1 person":  # "1 person" shouldn't be in the set anyway, but just to make sure
                write_read(f"{word}{'s' * (n > 1)}")
                time.sleep(1.5)
betterObjects.clear()   # clear the set
```

All of these additions and fixes to the code make my final project complete. My car was now able to move depending on the detections and display them properly on the LCD screen.

Results:

To test my project, I stood in front of the car and moved around to see if it followed me. I had to stand in a relatively blank area, as my car should only follow me if I'm the only thing it detects. My first attempt to test didn't work because YOLO kept identifying the project arm as a traffic light, in addition to recognizing myself as a human (at least the display worked properly–it showed "1 traffic light 1 person").

This video linked here shows the car turning and moving forward appropriately depending on where I stood and simultaneously the LCD display as it turns and moves. If it's waiting for a signal, it displays "looking…" If it's turning right or left, it will say so. If it's moving forward it will say, "Moving toward person". This second video shows what the YOLO program saw, including the output in the terminal in VS Code. This terminal output prints the objects it detects along with the x-coordinate of the center of the bounding box. Although it worked slowly, it still worked as desired and I'm very happy with the final product I made.

Analysis:

Overall, while my project did technically complete the basic goals I had in mind, it didn't work as well as I had hoped. For one, it moves extremely slowly. My initial vision for the project was to create a robot that was almost constantly moving, instantaneously following people around. The YOLO processes and sends signals so slowly that the car doesn't move smoothly. Some things that contributed to the slowness of it all was my laptop's graphics processing unit, or GPU. I don't think my laptop was powerful enough to fully support YOLO, let alone the YOLO plus the Arduino code. That's why in the second video, the camera output is extremely laggy. It looks like it's a little less than 1 frame per second, which is very bad for a video. Another reason why it was so slow was because of the timing issues between the Arduino and YOLO communication. I think this might be fixable by somehow altering the baud rates. I tried to increase the baud rate on the Arduino and the Python file, but then the display showed unreadable characters. I feel like there is a solution with the baud rates somehow, but I just didn't have enough time to fully understand and investigate how the baud rates work. This is one aspect of my project that could be improved.

Another thing that could be improved is the actual frame of the car. Besides the Arduino and the motors, I didn't secure the components on top of the car to the frame. If I had more time, I could've gotten velcro and neatly organized the top of the car. Furthermore, although Zip Ties provided a quick and easy way to secure the motors to the frame, using screws would have secured the motors better.

Lastly, the project could be improved by training YOLO to perform better in the classroom specifically. The generic COCO database is trained to identify "generic" things such as cars, traffic lights, pizza, giraffes, airplanes, and boats. It's possible to train YOLO to identify things specific to objects that could be found in Davidson's classrooms, potentially even being able to identify different people.

Conclusion:

All in all, although different from my initial project proposal, I was able to create a car that follows a person, if and only if a person is the only detection it makes. In going through different iterations of the project, figuring out what was possible and what wasn't, I learned to adapt and create a final project that worked. Using YOLOv5 object detection, I could control the car and use the LCD screen to display messages about what it was doing with each new action and detection. This project could be expanded and upgraded for several different applications, basically anything that uses autonomous robots and makes decisions based on its observations.