AI PROJECTS

Technical school Pirot

KA220-VET - Cooperation partnerships in vocational education and training

Project Title: AI tools for VET schools

Document Date: March 2026

This material has been compiled and prepared for purposes of Erasmus project by:
Technical school Pirot (author: Bojan Ćirić, co-authors:Boban Blagojević, Aleksandar Madić)
ICEP (author: Ladislav Mariš, co-author: Adelaida Fanfarova)
Agrupamento de Escolas Tomas Cabreira (author: Sandra Nobre, co-authors: Rui Dias, Gilherme Mota, Carla Lima)
Translated by: Bojana Stojanović

ADDRESS INFORMATION

Takovska 22, Pirot, Serbia
Web: https://book.tsp.edu.rs

Contents

PiRacer 3

1. PI-Racer Deep Learning Autopilot 3
   - 1.1 PiRacer Assembly Manual 3
   - 1.2 Raspian Legacy (Buster) Desktop 15
   - 1.3 WaveShare Branch for DonkeyCar Software 18
   - 1.4 Getting Started with DonkeyCar 19
2. Raspberry Pi remote access using RealVNC 21
   - 2.1 Enable VNC in Raspian OS with a or b: 21
   - 2.2 Install a VNC Viewer 22
3. Start Driving and Collect Data 23
4. Train Data - Create Inference Model 25
5. Load Model and Drive Autonomously 26

AR/VR APP DEVELOPMENT 29

1. Introduction 29

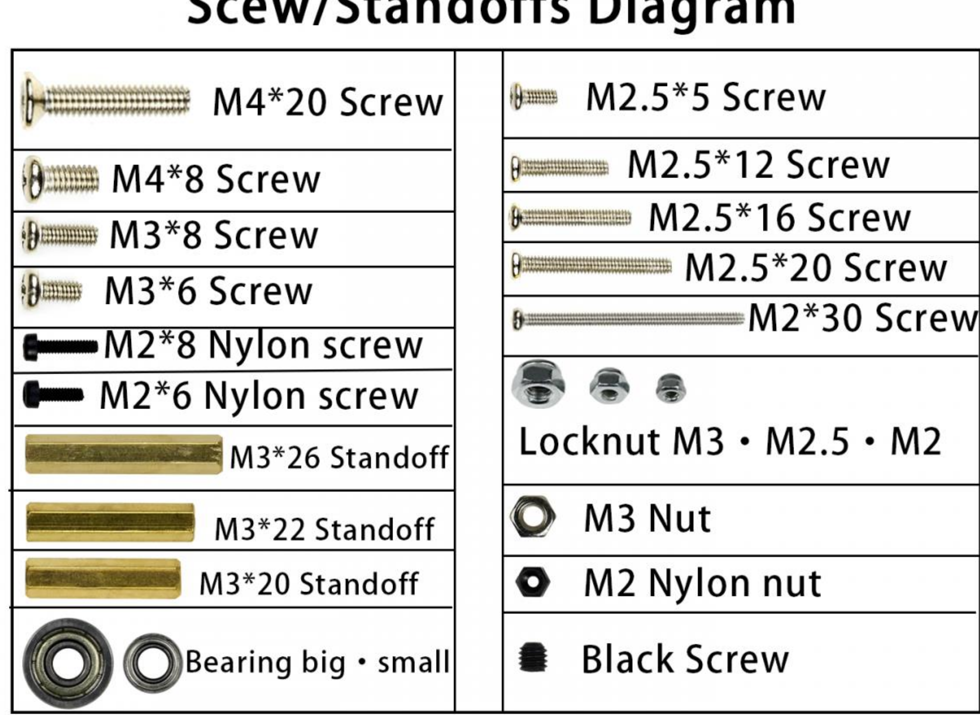
PiRacer

1. PI-RACER DEEP LEARNING AUTOPILOT

1.1 PiRacer Assembly Manual

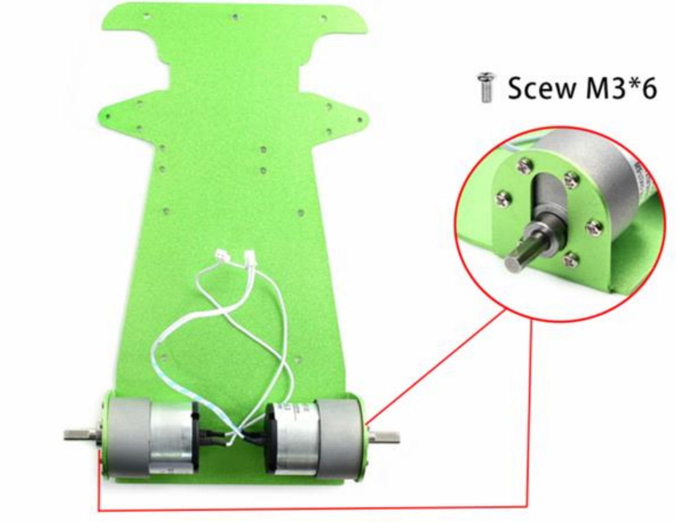
Screws/standoffs diagram

Diagram for reference. Note that the screws that come with servo wheel are not listed here.



1. Attach the motors to metal chassis

Note: Do not use the M3*8. It is longer and might damage the motor.

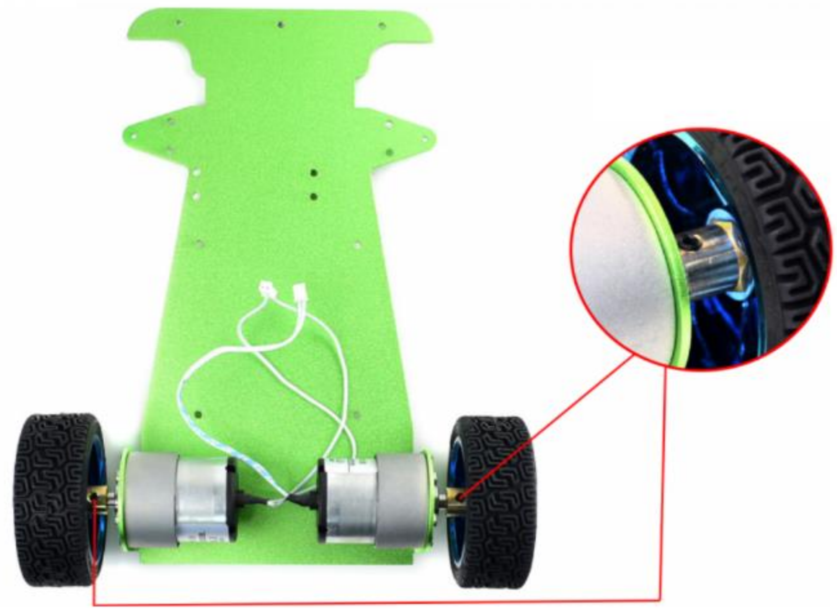


2. Add couplers to wheels

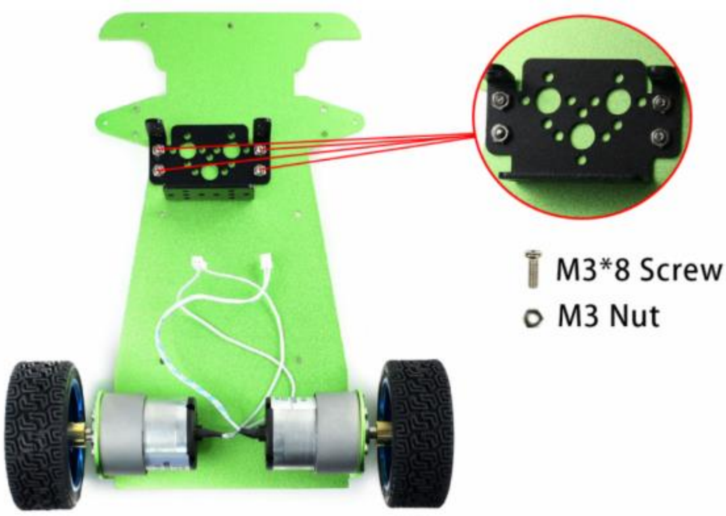
First, insert the black screw into the coupler. Then add the coupler to the wheel. You may need to press the coupler into the wheel. Secure the coupler to the wheel using M4*8 screw.

3. Assemble the wheels

Tighten the black screw to secure the coupler to the flat side of the axel

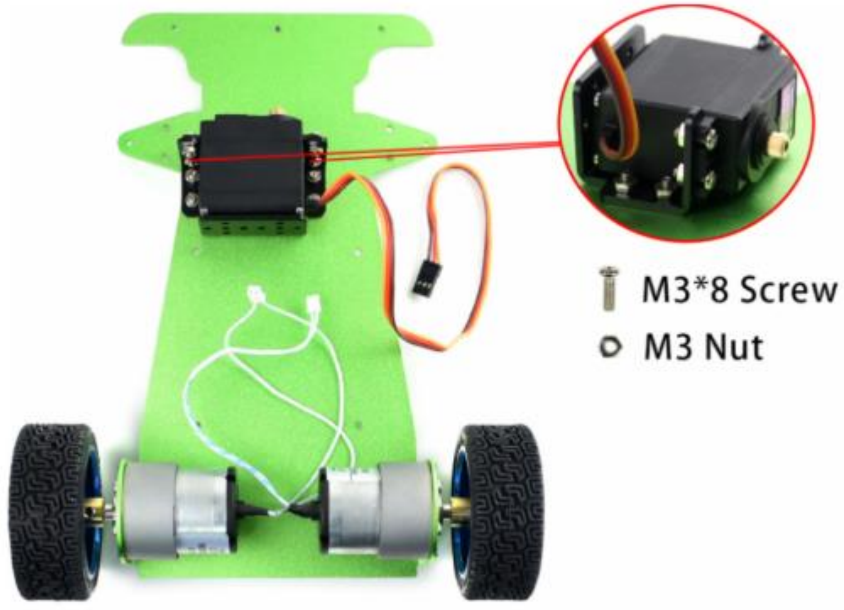


4. Mount the servo holder on metal chassis



5. Attach the servo on the holder and using the screws and nuts

Make sure the servo is in correctly. The outer shaft should be in the center.

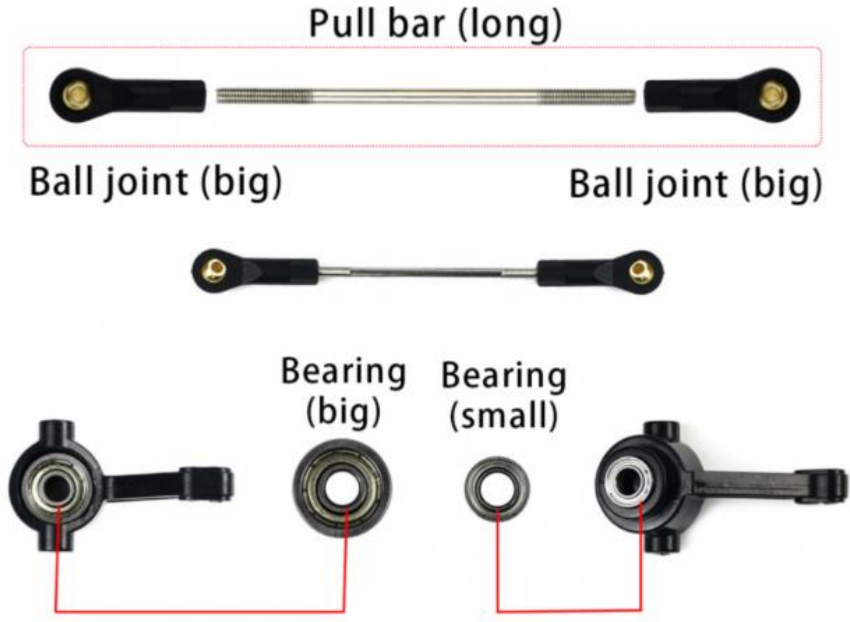


6. Assemble the servo pull bar

The pull bar is combined by using two ball joints, one flat and one ball, and the short bar. The two ball joints should be perpendicular to each other. Attach the servo wheel (horn) to the servo wheel holder using the screws that came with the servo wheel. Then attach flat ball joint to the servo wheel holder. Note that the groove of the servo wheel is toward outside.

7. Assemble the front-wheel pull bar and the steering knuckles

The front-wheel pull bar is combined by using two ball joints and the long bar. Then put the bearings in the steering knuckles. Each knuckle needs a small and a large bearing.



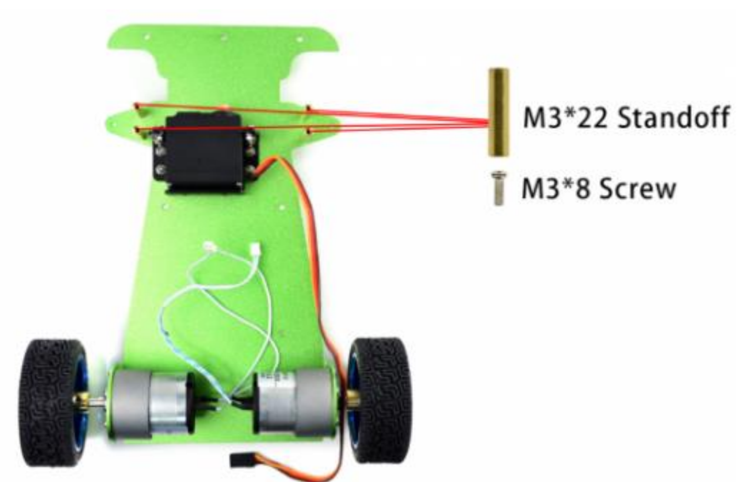
8. Assemble the servo pull bard, front-wheel pull bard, and the steering knuckles

Attach the servo pull bar on the top, and then the front-wheel pull bar, and finally the knuckles. The larger bearing should towards the inside.

9. Attach the wheels on the steering knuckle

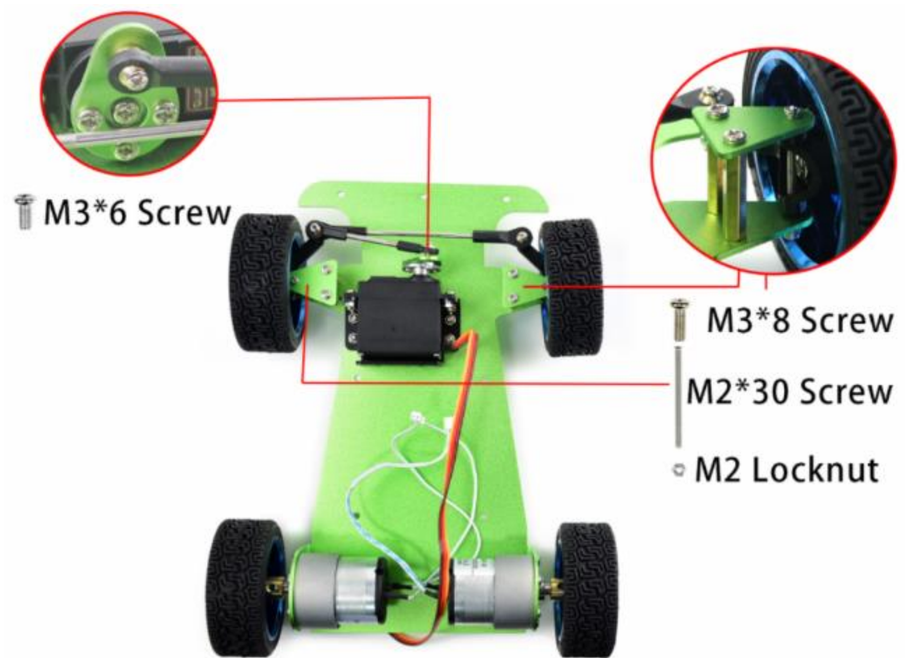
The nut should be on the outer side of the wheel, opposite the knuckle. Make sure the wheel is not too tight or too loose. Test the wheel to make sure it can move freely.

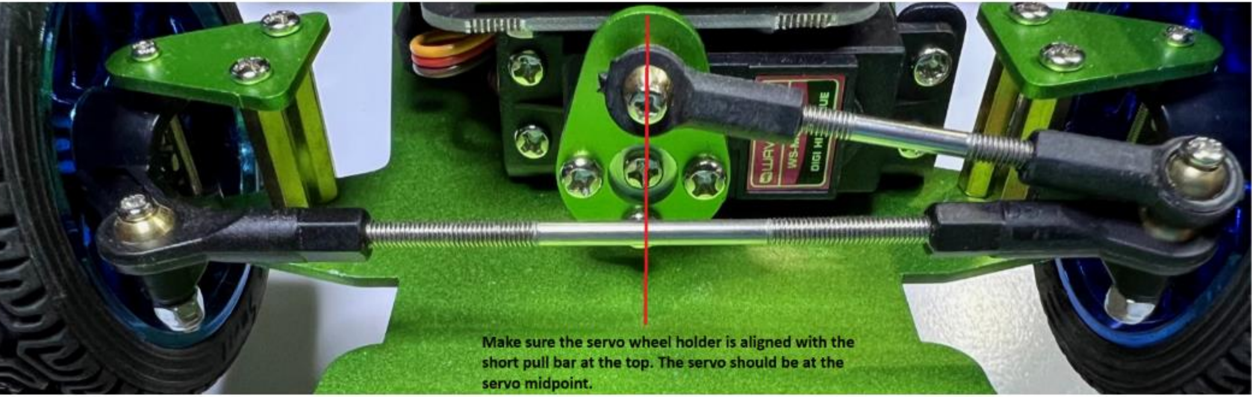
10. Add the M3 standoffs for the front wheels



11. Assemble the front-wheels combination

Put the servo wheel to servo, fix it by M3 screw. Fix the wheels by M2 screws and locknut and the triangle board. The front wheels should be straight forward. Adjust the long pull bar if needed.

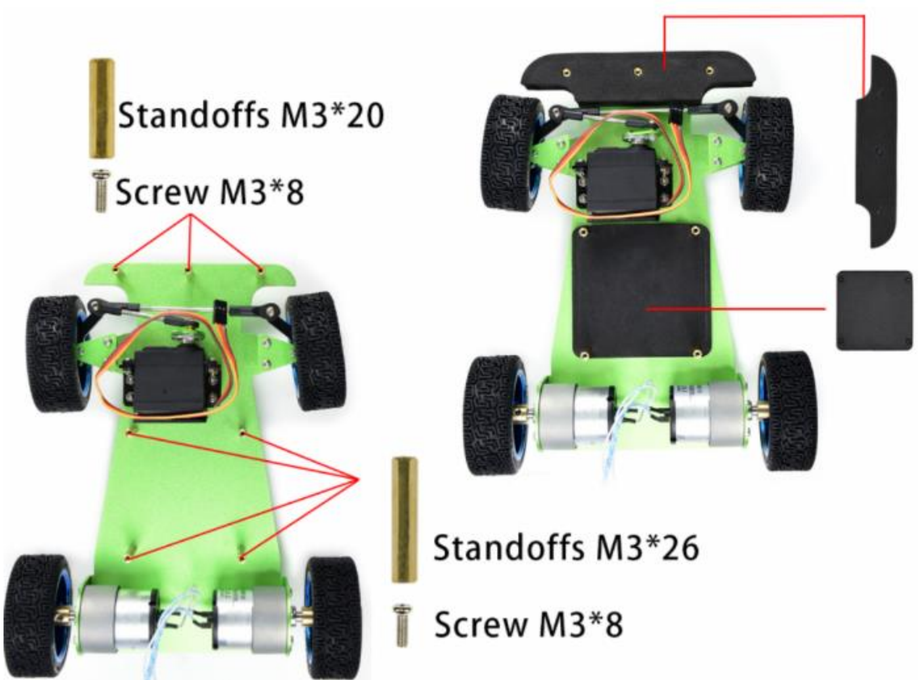




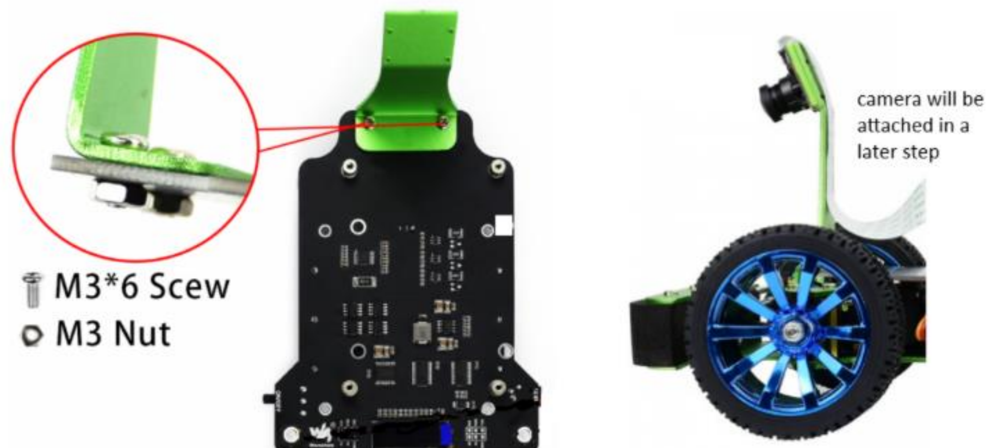
The front wheels should be straight forward. Adjust the long pull bar if needed.

12. Add the standoffs for PiRacer Expansion board and the bumper

Insert the EVA felt pads for the bumper and the PiRacer Expansion board.

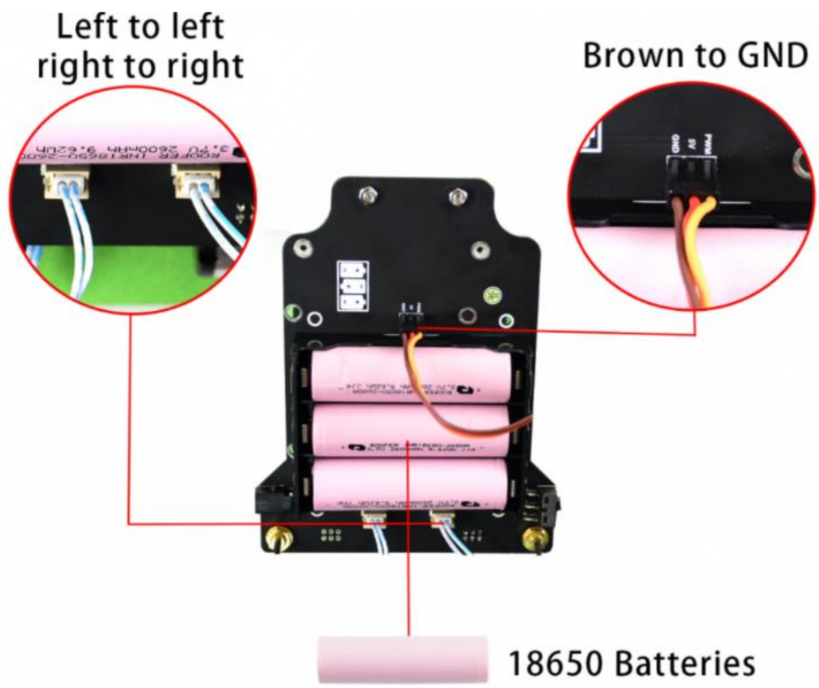


13. Attach the camera holder to PiRacer Expansion board.



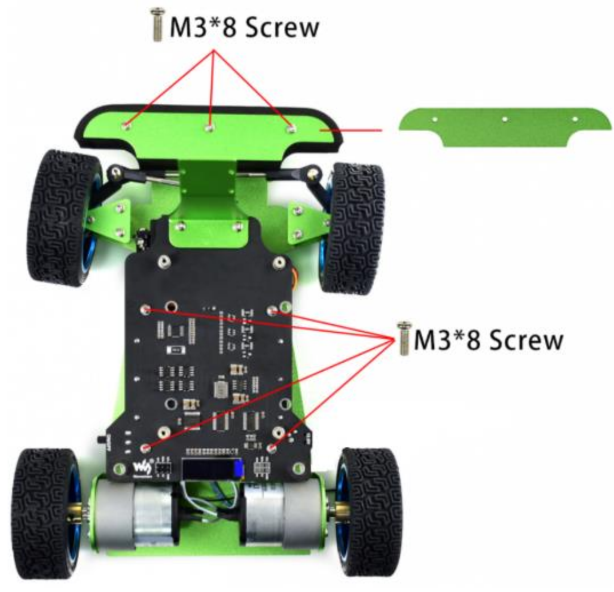
14. Insert the batteries in the correct direction

Connect the wires of the motors and servo to PiRacer Expansion board.



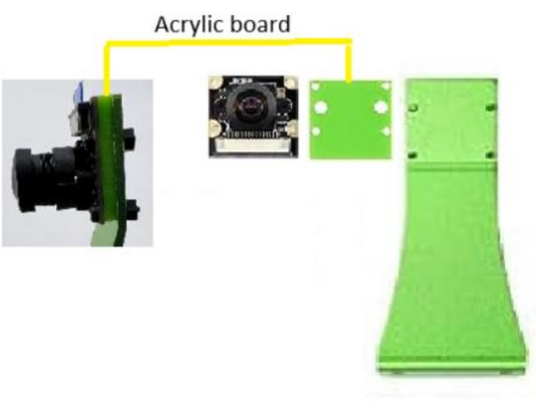
15. Attach the Expansion board and the metal bumper

Adjust the placement of motor and servo wires and attach the PiRacer Expansion board to the metal chassis and attach the metal bumper.

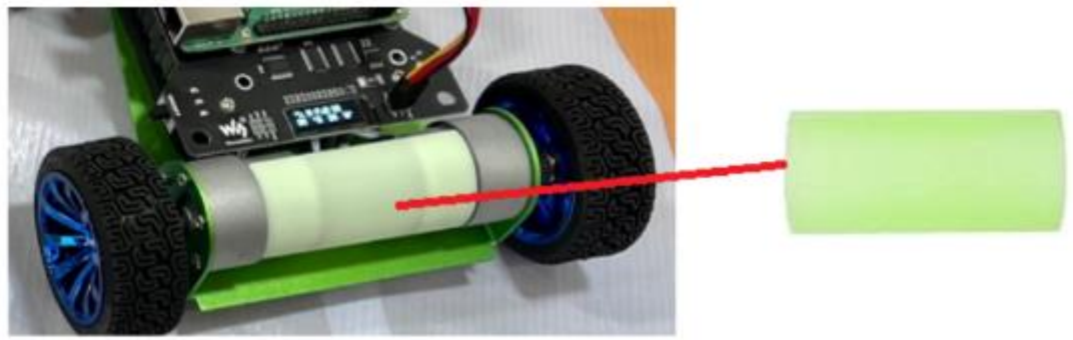


16. Mount camera to its holder using the nylon screws

Note: The Acrylic board should be between camera and the metal holder to avoid shorting.



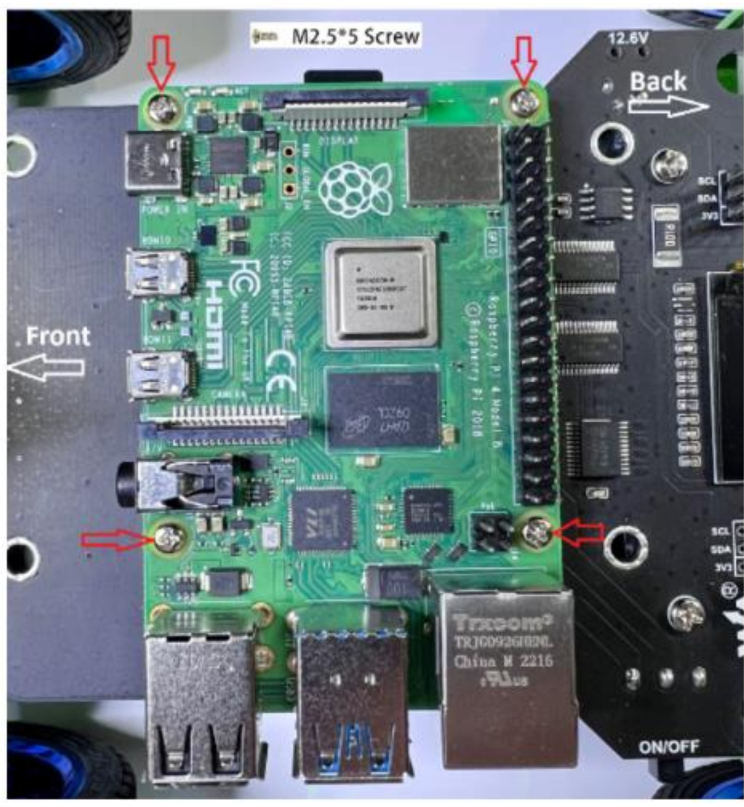
17. Attach the 3D-printed motor enclosure on DC motors



18. Power off the Raspberry Pi and disconnect the charger from the PiRacer Expansion board before proceeding.

19. Attach the Raspberry Pi to the PiRacer Expansion board

GPIO pins should be at the back of the car.

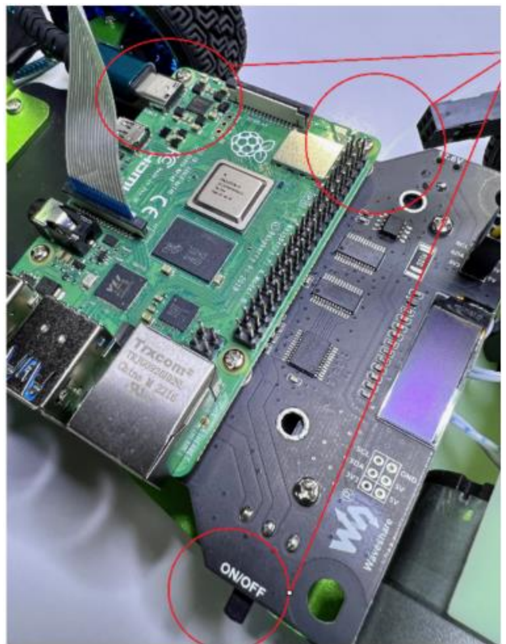


21. Attach the camera ribbon cable to the Raspberry Pi camera input

Blue side towards the Pi's USB ports.

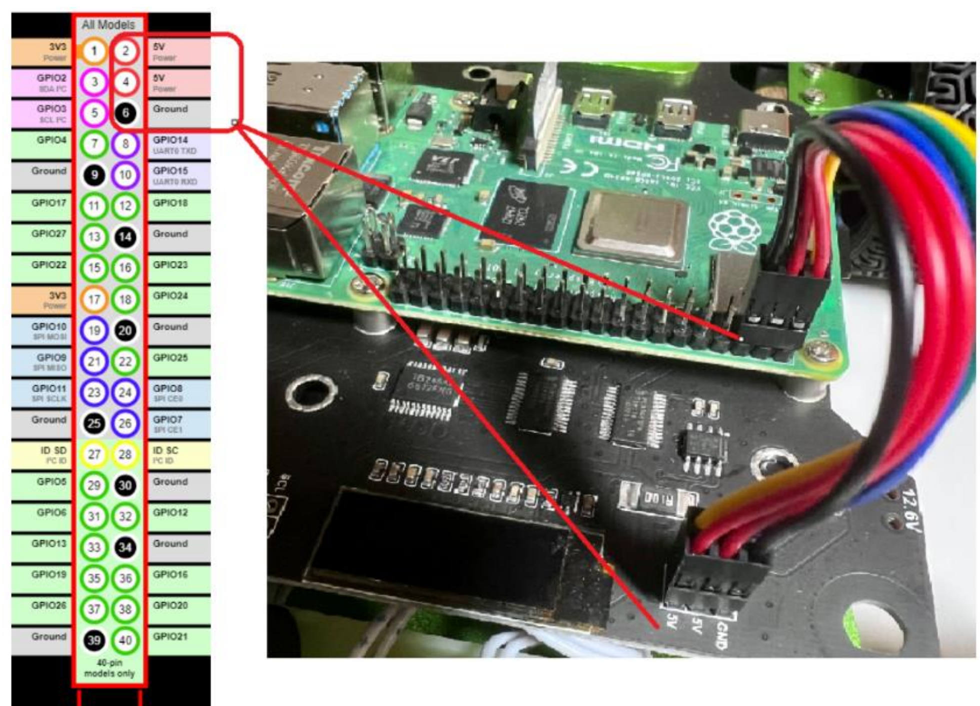
****WARNING ** Make sure the Raspberry Pi is not powered when connecting the 6 pin wires**

Also do not power the Raspberry Pi through the USB-C (external power) while it is powered by the PiRacer Expansion board.



22. Connect the Raspberry Pi to the PiRacer Expansion board using the 6PIN wires

Match the red|red|black (5V|5V|ground) on the Pi GPIO and the black|red|red (gnd|5V|5V) on the PiRacer Expansion board.



1.2 Raspian Legacy (Buster) Desktop

Flash OS - Raspian Legacy (Buster) Desktop

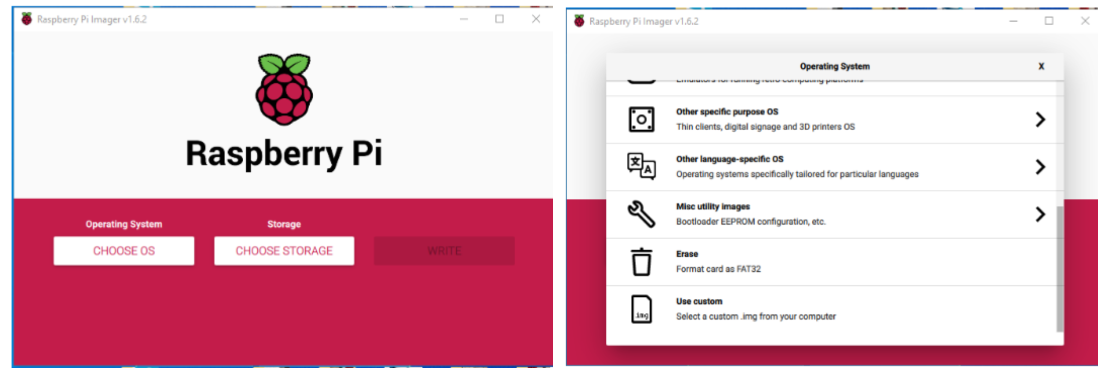
Navigate to the official download repository for Buster OS. **All earlier versions of Raspberry Pi OS can be found and downloaded here** and the directly previous Raspberry Pi 'Buster' OS **official download link is here**. See below for what that looks like, download the image file by clicking on the highlighted file.

Index of /raspios_armhf/images/raspios_armhf-2021-05-28

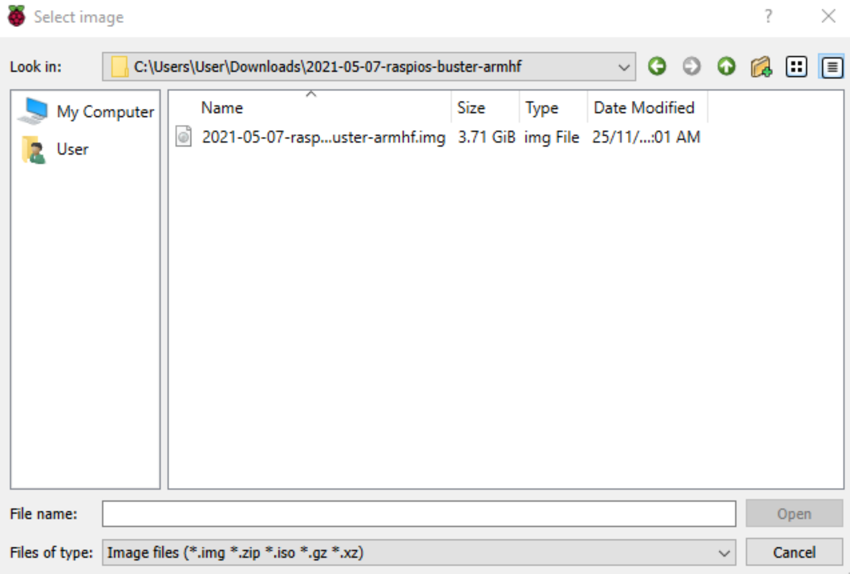
Name	Last modified	Size	Description
Parent Directory		-	
2021-05-07-raspios-buster-armhf.info	2021-05-07 16:07	188K	
2021-05-07-raspios-buster-armhf.zip	2021-05-07 16:12	1.2G	
2021-05-07-raspios-buster-armhf.zip.sha1	2021-05-28 15:45	78	
2021-05-07-raspios-buster-armhf.zip.sha256	2021-05-28 15:45	102	
2021-05-07-raspios-buster-armhf.zip.sig	2021-05-28 15:00	488	
2021-05-07-raspios-buster-armhf.zip.torrent	2021-05-28 15:45	23K	

Download and Install Raspberry Pi Imager from https://www.raspberrypi.com/software/

Now, let's open up the Official Raspberry Pi Imager, see it in the image below. Worth noting here - if you hit **| CTRL+SHIFT+X |** on your keyboard while in the Raspberry Pi Imager Program it will open the Advanced Hidden Menu. This Hidden Menu allows you to preconfigure your Raspberry Pi with SSH, WIFI credentials, and Localisation settings.



Having done this it will open up a file explorer. Navigate to that extracted Disc Image File and click on it. See this in the image below. Then open the file image and this will arm the Raspberry Pi Imager with the Raspberry Pi 'Buster' OS. Find it, select it, and open it.



Insert the Micro-SD card that you want to be flashed into your computer. Use a **USB to Micro-SD adapter** if needed. Then click on the **| CHOOSE STORAGE |** button and select your inserted Micro-SD. Keep in mind any data that was on your Micro-SD card will be wiped/permanently deleted when it is flashed. The Official Raspberry Pi Imager will now look like the image below.

With everything sorted (the right OS loaded and the correct Storage selected) you can now click the **| WRITE |** button to start the flashing process. See this flashing process in the image below.

Once the flash is complete it will automatically virtually eject the Micro-SD card from the computer. So then you can simply physically take out your Micro-SD and insert it into a Raspberry Pi single-board computer. Then set your Raspberry Pi up normally as a desktop computer. Once it boots you will be greeted by the old familiar background, see image below, and you will have successfully flashed 'Buster' OS to your Raspberry Pi. You are free now to roam your well-acquainted digital ground.

Follow Pi Wizard instruction to set location, keyboard, wifi, and get update

Menu / Preferences / Raspberry Pi Configuration/ Interfaces

- enable Camera and I2C
- optional: enable VNC for remote access
- Click OK and reboot

Additional Software

Java 3.8.3 (run one at a time)

```
sudo apt install build-essential libncurses5-dev libgdbm-dev libnss3-dev libssl-dev
libreadline-dev libffi-dev -y
wget https://www.python.org/ftp/python/3.8.3/Python-3.8.3.tgz
tar -zxvf Python-3.8.3.tgz
cd Python-3.8.3
sudo ./configure --enable-optimizations
sudo make -j 4
sudo make altinstall
sudo python3.8 -m pip install boto3
sudo python3.8 -m pip install tqdm
```

Additional tools

```
sudo apt install chromium-browser -y
sudo apt-get install zip unzip -y
```

AWS CLI

Default user accou nt on the Raspberry Pi

```
sudo apt install awscli -y
sudo pip3 install --upgrade awscli
sudo pip3 install boto3

user - pi
pw - raspberry
```

1.3 WaveShare Branch for DonkeyCar Software

Install Dependencies

```
sudo apt-get install build-essential python3 python3-dev python3-pip
python3-virtualenv python3-numpy python3-picamera python3-pandas
python3-rpi.gpio i2c-tools avahi-utils joystick libopenjp2-7-dev
libtiff5-dev gfortran libatlas-base-dev libopenblas-dev
libhdf5-serial-dev git ntp -y
```

Optional?

```
sudo apt-get install libilmbase-dev libopenexr-dev libgstreamer1.0-dev
libjasper-dev libwebp-dev libatlas-base-dev libavcodec-dev libavformat-dev
libswscale-dev libqtgui4 libqt4-test -y
```

Setup Virtual Environment

```
python3 -m virtualenv -p python3 env --system-site-packages
echo "source ~/env/bin/activate" >> ~/.bashrc
source ~/.bashrc
```

Install DonkeyCar Python Code - WaveShare Branch for DonkeyCar Software 3.1.0

```
mkdir projects
cd projects
git clone https://github.com/waveshare/donkeycar

cd donkeycar
git checkout master
pip install -e .[pi]

pip install tensorflow==1.13.1
```

Not needed pip install numpy --upgrade

```
pip install protobuf==3.20.*
```

Test tensorflow version - should show 1.13.1

```
python -c "import tensorflow; print(tensorflow.__version__)"
```

NOTE: Could not run model on car until running the following two

```
pip install https://github.com/lhelontra/tensorflow-on-arm/releases/download/v2.2.0/
        tensorflow-2.2.0-cp37-none-linux_armv7l.whl

pip install tensorflow==1.13.1
```

Edit camera.py to add self.camera.hflip = True

```
sudo nano /home/pi/projects/donkeycar/donkeycar/parts/camera.py
```

```
class PiCamera(BaseCamera):
    def __init__(self, image_w=160, image_h=120, image_d=3, framerate=20):
        from picamera.array import PiRGBArray
        from picamera import PiCamera

        resolution = (image_w, image_h)
        # initialize the camera and stream
        self.camera = PiCamera() #PiCamera gets resolution (height, width)
        self.camera.vflip = True
        self.camera.hflip = True
        self.camera.resolution = resolution
        self.camera.framerate = framerate
        self.rawCapture = PiRGBArray(self.camera, size=resolution)
        self.stream = self.camera.capture_continuous(self.rawCapture,
            format="rgb", use_video_port=True)

        # initialize the frame and the variable used to indicate
        # if the thread should be stopped
```

Optional Install OpenCV

```
sudo apt install python3-opencv -y
```

Test with:

python -c "import cv2"

1.4 Getting Started with DonkeyCar

Open a terminal window enter the following command

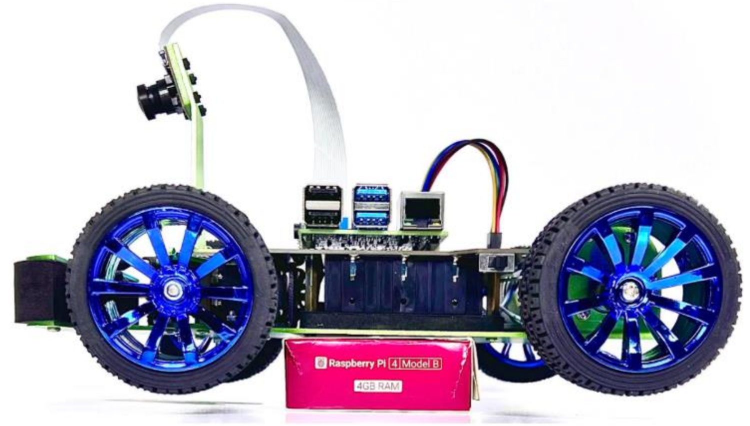
Create DonkeyCar App

This will create a folder called mycar with all the python code needed to drive the car.

Calibrate the front steering

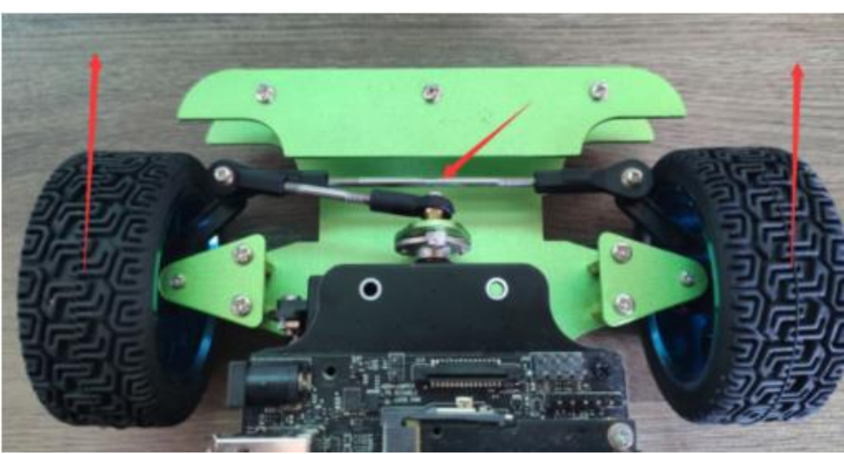
Make sure your car is off the ground to prevent a runaway situation.

Use a small box like the on the Raspberry Pi came in.

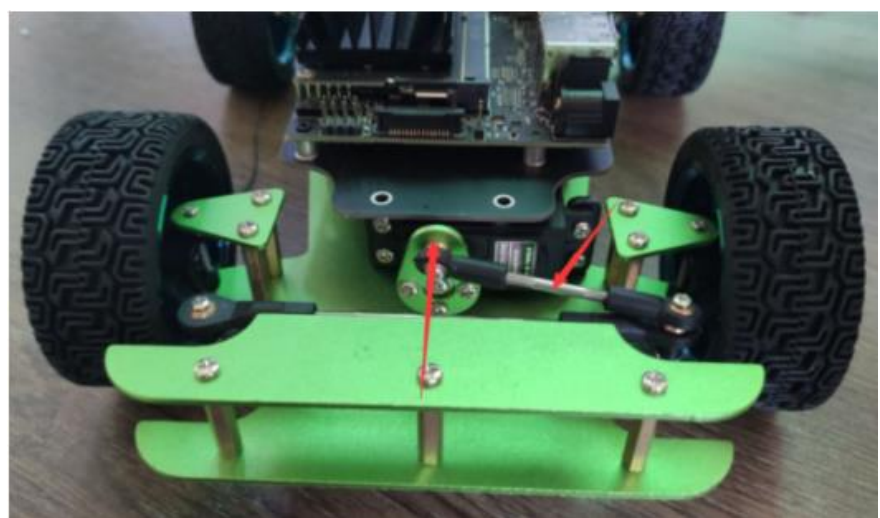


To make sure that the DonkeyCar can drive straight and make the needed turns on the track, the car's hardware and software need to be calibrated.

The front wheels should be straight forward. Adjust the long pull bar if needed



The servo wheel holder should be aligned with the short pull bar at the top. Adjust the short pull bar if needed.



Calibrate the servo software

Find the the left, center and right steering PWM for this car's servo.

The center should be half way between the left and right. example:

Left 200
right 560
center will be 380

In a terminal window enter the following

```
cd ~/mycar
donkey calibrate --channel 0 --bus=1
```

Try values 300, 400, 500 and see how the steering changes. Figure out the max turn for left and right.

Once you have the numbers, edit the config.py and update the values you found.

The throttle should be set using your numbers for the steering. The throttle is already set correctly.

nano config.py

```
#STEERING
STEERING_CHANNEL = 0            #channel on the 9685 pwm board 0-15
STEERING_LEFT_PWM = 200         #pwm value for full left steering
STEERING_RIGHT_PWM = 560        #pwm value for full right steering

#THROTTLE
THROTTLE_CHANNEL = 0            #channel on the 9685 pwm board 0-15
THROTTLE_FORWARD_PWM = 4095     #pwm value for max forward throttle
THROTTLE_STOPPED_PWM = 0        #pwm value for no movement
THROTTLE_REVERSE_PWM = -4095    #pwm value for max reverse throttle
```

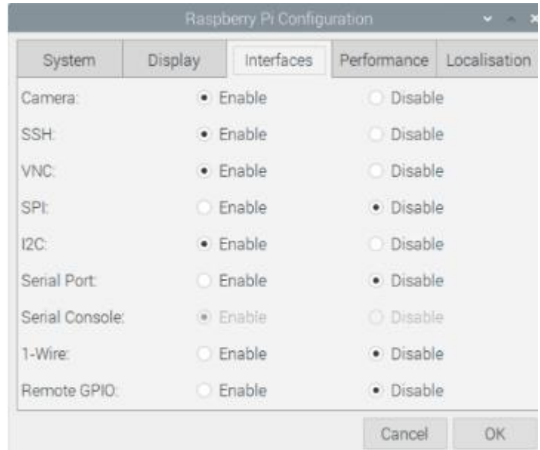
Install OLED Display Service

```
cd ~
git clone https://github.com/waveshare/pi-display
cd pi-display
sudo ./install.sh
cd ~
```

2. RASPBERRY PI REMOTE ACCESS USING REALVNC

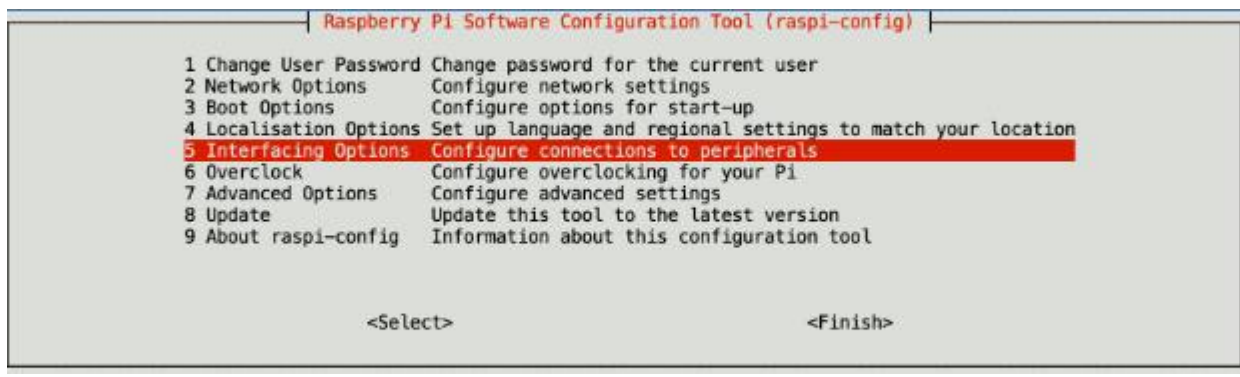
2.1 Enable VNC in Raspian OS with a or b:

a) Enable VNC in Preferences - Raspberry PI Configuration – Interfaces

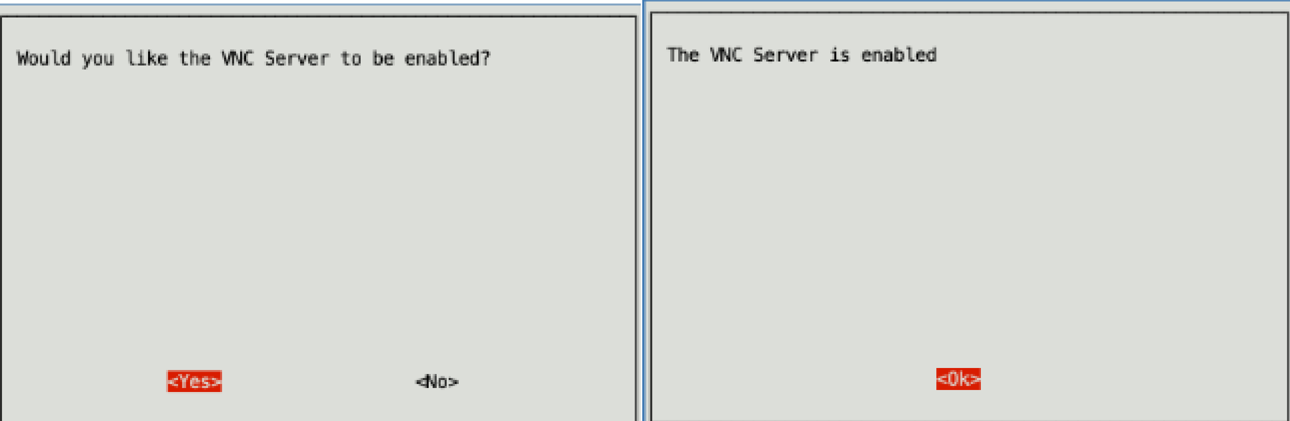


b) Another method to enable VNC is using Terminal, enter the command

```
sudo raspi-config
```



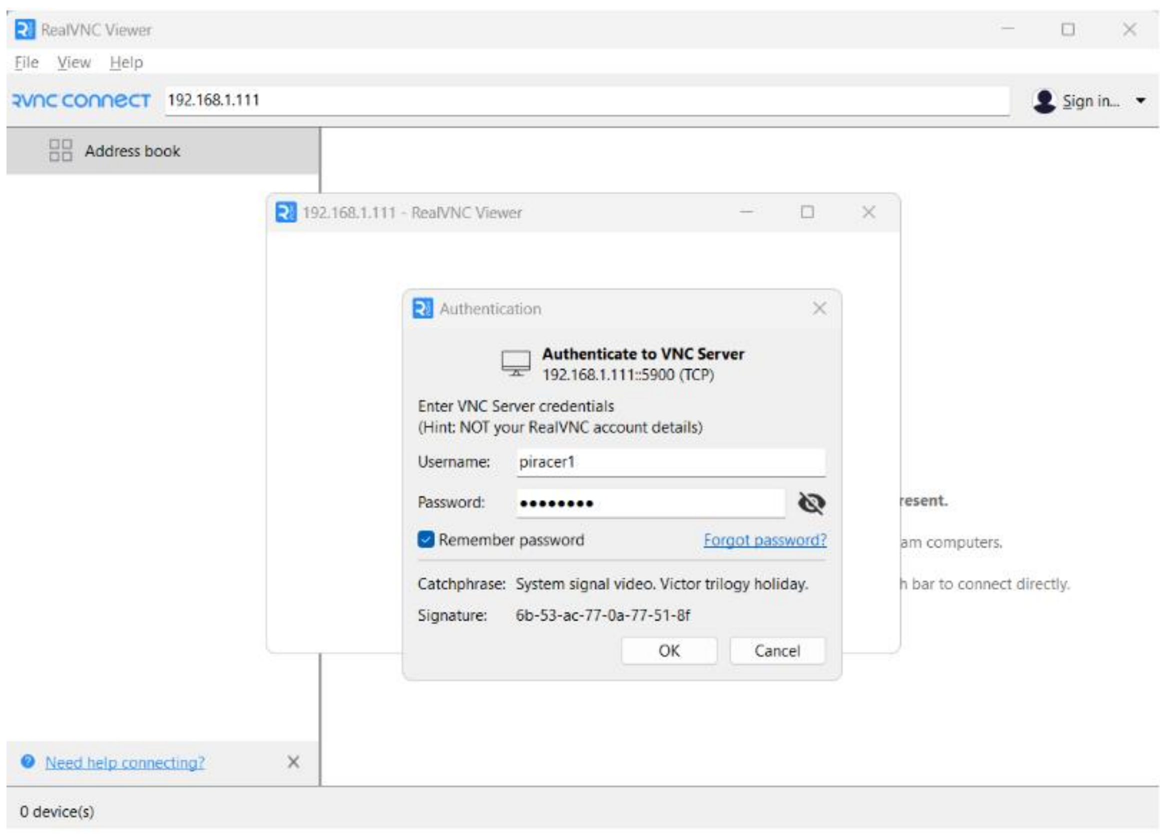

```
Raspberry Pi Software Configuration Tool (raspi-config)

P1 Camera      Enable/Disable connection to the Raspberry Pi Camera
P2 SSH         Enable/Disable remote command line access to your Pi using SSH
P3 VNC         Enable/Disable graphical remote access to your Pi using RealVNC
P4 SPI         Enable/Disable automatic loading of SPI kernel module
P5 I2C         Enable/Disable automatic loading of I2C kernel module
P6 Serial      Enable/Disable shell and kernel messages on the serial connection
P7 1-Wire      Enable/Disable one-wire interface
P8 Remote GPIO Enable/Disable remote access to GPIO pins

<Select>                                   <Back>
```



2.2 Install a VNC Viewer

You will need to install a VNC Viewer on your computer, so you can connect to you Raspberry Pi. There are a number of viewers available, but the easiest to set up is **Real VNC Viewer**. You can download Windows and Mac installers from here: https://www.realvnc.com/en/connect/download/viewer/

Open Real VNC Viewer and enter Piracer IP adress (read it from car display)



3. START DRIVING AND COLLECT DATA

Start your car

```
cd ~/mycar
python manage.py drive
```

This script will start the drive loop in your car which includes a part that is a web server for you to control your car. You can now control your car from a web browser at the URL: <your car's hostname.local>:8887

Open a browser and connect to the DonkeyCar Monitor at **localhost:8887**

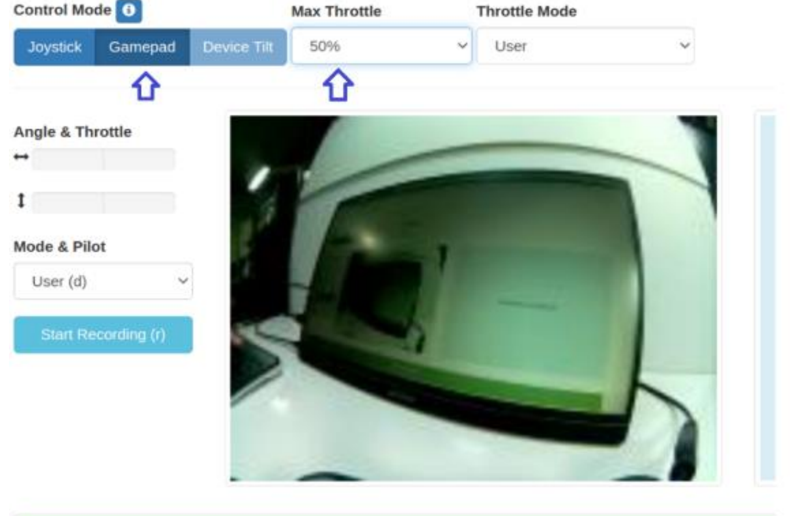
NOTE: When the car is started, a folder created under /mycar/data called tub_#_date to store the data for the session. The data is gathered when the throttle is engaged. To gather a clean dataset, best practice is to stop the "python manage.py drive" using **Contole + C** and restart to begin the training with a new /mycar/data/tub.

There are 2 drive options

1. Use the gamepad (recommended)

After starting the car, open a web browser from the Raspberry Pi desktop while the car is connected to the monitor.

Select Gamepad and set the throttle to 50% to start practicing. Test turns and speed while the car is on the Pi box.



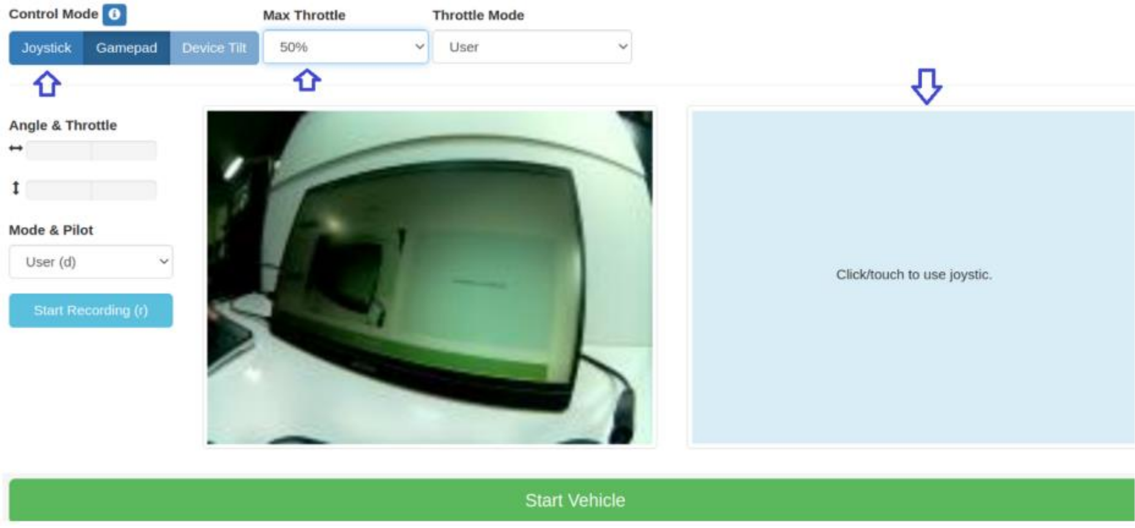
When you are ready to drive on the track, disconnect the monitor, place the car on the track and start driving. Start slow until you are comfortable handling the car.

2. Use the web browser joystick

You will need to use a tablet or phone connected to the same wifi. After starting the car, use a tablet or phone to connect to the car's web server using the car's IP:8887. The car should display its IP if the OLED Display Service was enabled in a previous step.

Select Joystick (the joystick control area is labeled **click/touch to use joystic**) Use the joystick area to drive the car with your finger.

© 2026 https://book.tsp.edu.rs



When finished driving, stop the "python manage.py drive" using **Contole + C**.

4. TRAIN DATA - CREATE INFERENCE MODEL

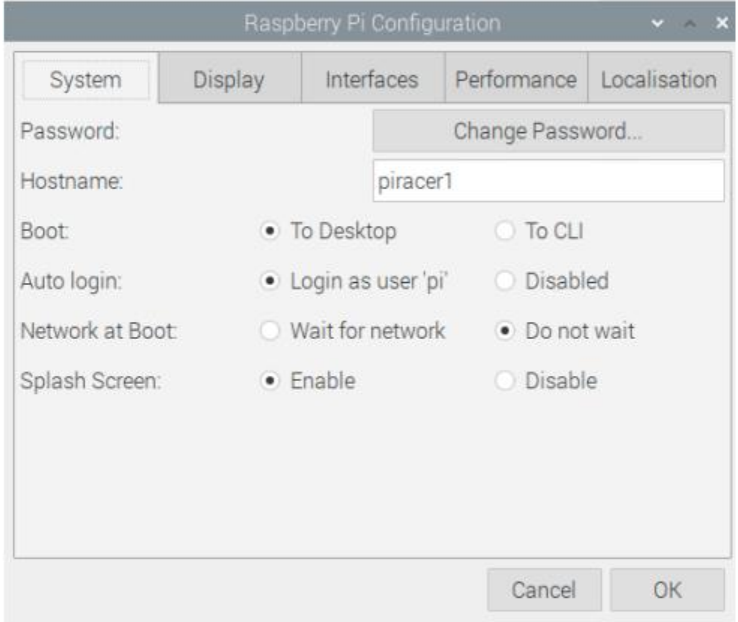
Data location is /home/pi/mycar/data. All tub folders in this directory will be used to create your model. Remove anything you don't want included. You can train data with Raspberry Pi or with AWS virtual machine .

1. Train with Raspberry Pi

Train Data:
In a new terminal session on your host PC use rsync to copy your cars folder from the Raspberry Pi

```
rsync -rv --progress --partial <hostname>@<your_pi_ip_address>:~/mycar/data/
~/mycar/data/
etc:
rsync -rv --progress --partial piracer1@192.168.1.111:~/mycar/data/  ~/mycar/data/
```

You can find hostname in Raspberry Pi Configuration:



Train model:
In the same terminal you can now run the training script on the latest tub by passing the path to that tub as an argument. You can optionally pass path masks, such as ./data/* or ./data/tub_?_17-08-28 to gather multiple tubs. For example:

```
python ~/mycar/manage.py train --tub <tub folder names comma separated> --model
./models/mypilot.h5
etc:
python ~/mycar/manage.py train --tub ./data/tub_1_24-04-06 --model
./models/mypilot.h5
```

It will cost a long time to train a model, please be patient.
After training, you can get a model, you need to copy the module to your raspberry Pi and test it.

```
rsync -rv --show-progress --partial ~/mycar/models/
<hostname>@<your_ip_address>:~/mycar/models/
etc:
rsync -rv --show-progress --partial ~/mycar/models/
piracer1@192.168.1.111:~/mycar/models/
```

5. LOAD MODEL AND DRIVE AUTONOMOUSLY

Start your car with the model

```
cd ~/mycar
python manage.py drive --model ~/mycar/models/mypilot.h5
```

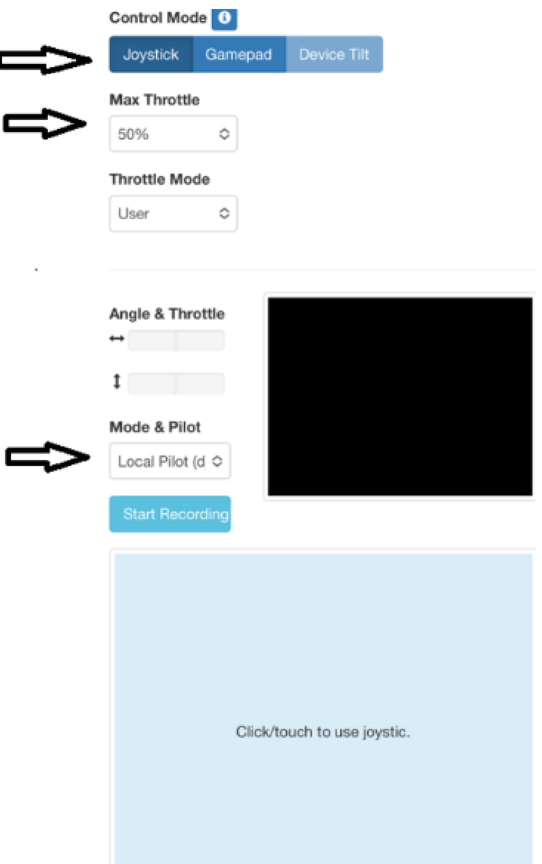
For autonomous driving or steering, use the web browser joystick on another device.

You will need to use a tablet or phone connected to the same wifi. After starting the car, use a tablet or phone to connect to the car's web server using the car's IP:8887. The car should display its IP if the OLED Display Service was enabled in a previous step.

Autonomous Driving

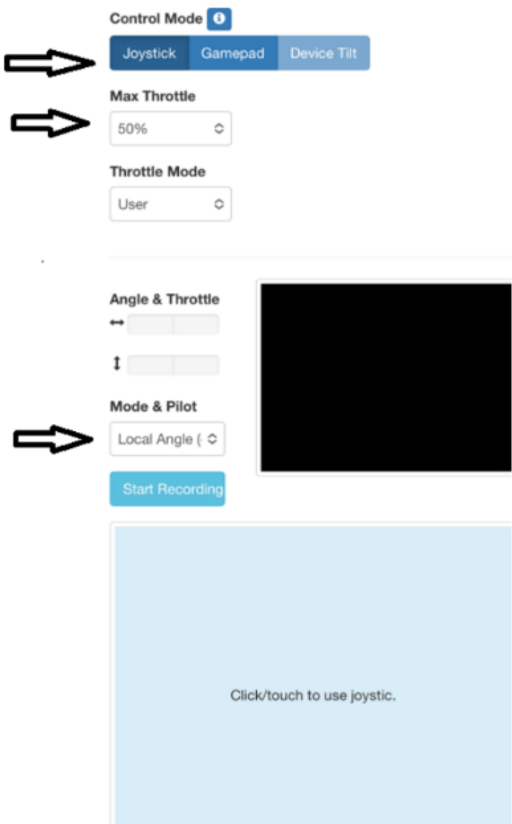
NOTE: As soon as **Local Pilot** is selected the car will start autonomous driving

- **Control Mode:** Select **Joystick ** (the joystick control area is labeled **click/touch to use joystic**)
- **Max Throttle:** Set the Max throttle to about 50% to keep the car from going too fast
- **Mode & Pilot:** Select Local Pilot (Local Pilot is autonomous)



Autonomous Steering

- **Control Mode:** Select **Joystick ** (the joystick control area is labeled **click/touch to use joystic**)
- **Max Throttle:** Set the Max throttle to about 50% to keep the car from going too fast
- **Mode & Pilot:** Select Local Angle (Local Angle is the car steering while you provide the throttle)



Faster autonomous model?

Use your working autonomous driving model to gather new data at a faster speed. Use Local Angle so the car steers while you provide faster throttle.

© 2026 https://book.tsp.edu.rs

AR/VR APP DEVELOPMENT

1. INTRODUCTION

The Meta Quest 3 marks a pivotal shift from pure Virtual Reality (VR) to high-fidelity Mixed Reality (MR). By leveraging high-resolution color passthrough and a significantly more powerful Snapdragon XR2 Gen 2 chipset, it allows digital content to coexist seamlessly with the physical world. With its slimmer "pancake" optics and 4K+ Infinite Display, it is currently the most accessible powerhouse for both consumers and developers.

1.1 Introduction to Meta Quest 3 and Mixed Reality

The Meta Quest 3 utilizes high-resolution Color Passthrough technology. Unlike traditional VR, where the world is entirely virtual, **Mixed Reality (MR)** uses onboard cameras to project the real environment and then overlays **WebGL** elements on top. Your code utilizes the immersive-ar mode, which is the core of this spatial experience.

System Architecture (The Web Stack)

The application functions as a "sandwich" of layers:

- **Layer 1 (Hardware)**: Quest 3 sensors, depth projectors, and RGB cameras.
- **Layer 2 (Browser)**: Meta Quest Browser (Chromium-based with WebXR support).
- **Layer 3 (API)**: WebXR Device API, which acts as the bridge between hardware and code.
- **Layer 4 (Logic)**: Your JavaScript logic using Three.js for rendering and TensorFlow.js for vision.

2. WEBXR: THE BROWSER-BASED ALTERNATIVE

WebXR allows developers to create immersive experiences that run directly in the Meta Quest Browser. It is built on standard web technologies, making "VR on the web" as easy to access as a website.

Why Choose WebXR?

- Frictionless Access: Users don't need to download large files from the Meta Store; they simply click a URL and hit "Enter VR."
- Frameworks: It utilizes powerful JavaScript libraries like Three.js, A-Frame (HTML-based), or Babylon.js.
- Cross-Platform: A single WebXR app can often run on a Quest 3, an Android phone (AR mode), or a desktop PC.

2.1 Library Analysis

Three.js (The Render Engine)

Three.js handles the heavy lifting of WebGL. It manages the Scene, Camera, and Renderer. In your code, the renderer is set to alpha: true, which is vital for AR as it allows the "empty" parts of the browser to become a window into the real world.

TensorFlow.js & COCO-SSD

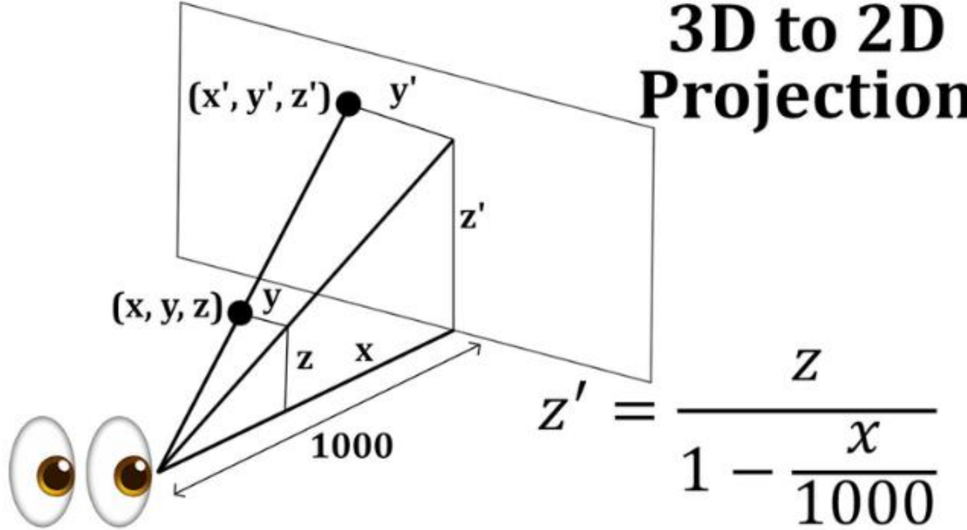
COCO-SSD (Common Objects in Context - Single Shot MultiBox Detector) is a model trained to recognize 80 classes of objects. The beauty of TF.js is its **WebGL backend**, which ensures that AI calculations are performed on the Quest's GPU rather than the CPU, preventing the device from overheating.

2.2 Mathematical Projection (2D to 3D)

This is the most technical part of the script. The AI model returns a bbox (bounding box) in pixels (e.g., x=100, y=200).

The detectionTo3D function performs un-projection:

- **Normalization**: It converts screen coordinates to a range of -1 to $+1$.
- **FOV Calculation**: It factors in the camera's Field of View.
- **Vector Ray**: It creates a direction vector from the user's head position toward the object.
- **Placement**: It places the 3D label on that vector at a defined distance (DIST).



2.3 Implementing Hit-Testing

Hit-testing allows the app to "fire" an invisible ray (raycast) into real space to detect intersections with floors or tables. When the ***hitTestSource*** returns a result, the cursor (green ring) is snapped to that pose (position and orientation).

2.4 AI Pipeline: Detection and Labeling

In the code, detection is not performed every frame (which would lag the headset), but every $2000ms$ (DETECT_INTERVAL_MS).

- **Label Sprite**: Since Three.js cannot render standard HTML fonts directly in a 3D scene, we use a CanvasTexture. We "draw" the text on an invisible 2D HTML canvas and apply it as a texture to a 3D Sprite that always faces the user (billboarding).

2.5 The Necessity of HTTPS

WebXR and camera access (getUserMedia) are classified as "Powerful Features" by browser vendors. They will only work in a **Secure** Context.

- **Localhost Exception**: You can test on your PC using http://localhost, but as soon as you try to access that server from your Quest 3 headset, the browser will block XR features because it sees an insecure network IP (e.g., http://192.168.1.10).

- **The Solution**: You must use a **Tunneling Service** or **Secure Hosting**.

2.6 Meta Quest Link & Developer Mode

To run and debug your code efficiently, your Quest 3 must be recognized as a developer device.

Step-by-Step Activation:

- **Developer Account**: Register at dashboard.oculus.com. You will need to create an "Organization" (it can be any name).

- Mobile App: Open the Meta Quest app on your phone, go to **Menu > Devices > Headset Settings > Developer Mode**, and toggle it **ON**.

- **The Link**: Connect your Quest 3 to your PC using a high-quality USB-C 3.0 cable or via Air Link (High-speed Wi-Fi 6).

2.7 The Three.js Foundation (The Boilerplate)

Before entering AR, we must initialize a standard 3D environment. However, for Quest 3, two settings are non-negotiable:

- **Alpha & Antialias**:

```
javascript
const renderer = new THREE.WebGLRenderer({ antialias: true, alpha: true });
```

- **XR Activation**:

```
enderer.xr.enabled = true;
```

This tells Three.js to listen for the Quest's head-tracking data ($6DOF$) and apply it to the `camera` object automatically.

2.8 Managing the AR Session (The Lifecycle)

The "Enter AR" button triggers **navigator.xr.requestSession**. This is where we define what "superpowers" our app needs from the Quest 3 hardware.

Required & Optional Features:

- **local-floor**: This tells the Quest to set the $Y=0$ coordinate at the actual physical floor level.
- **hit-test**: Enables the ability to raycast against real-world geometry.
- **plane-detection**: Requests the "Semantic" data (knowing which mesh is a 'table' vs. a 'wall').

```
JavaScript
const session = await navigator.xr.requestSession('immersive-ar', {
    requiredFeatures: ['local-floor'],
    optionalFeatures: ['hit-test', 'plane-detection']
});
```


© 2026 https://book.tsp.edu.rs

3. AR + AI OBJECT DETECTION DEMO

3.1 Project Overview

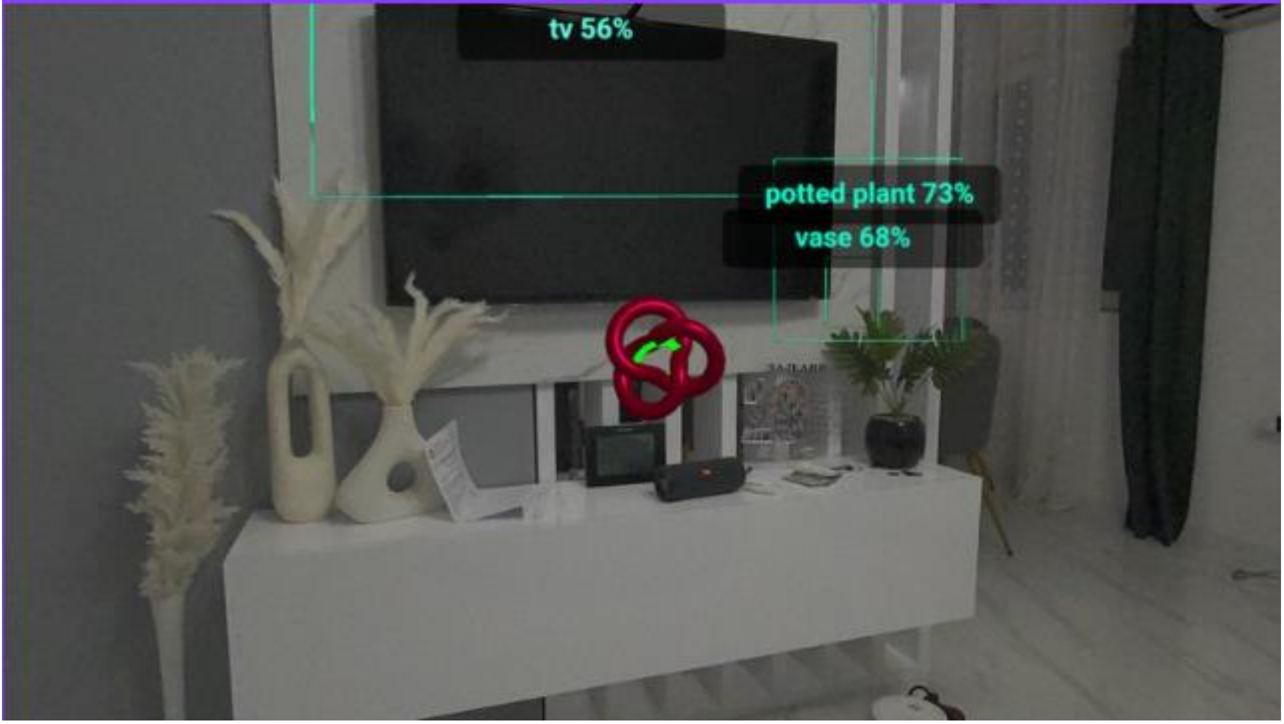
This is a WebXR Augmented Reality application for Meta Quest 3 that combines:

- **Passthrough AR** — the real room is visible through the headset cameras
- **Hit-test interaction** — a virtual cursor that snaps to real-world surfaces
- **AI object detection** — TensorFlow.js recognizes objects in the camera feed and displays labels in 3D space
- **Scene Understanding** — WebXR Plane Detection visualizes detected surfaces (floor, walls, table)

Live URL: http://92.113.18.92/

Single file:

index.html



3.2 Project Architecture

🗎 index.html (all-in-one)

│

├── HTML → canvas + UI button + hidden video element

├── CSS → transparent background, overlay UI

└── JavaScript

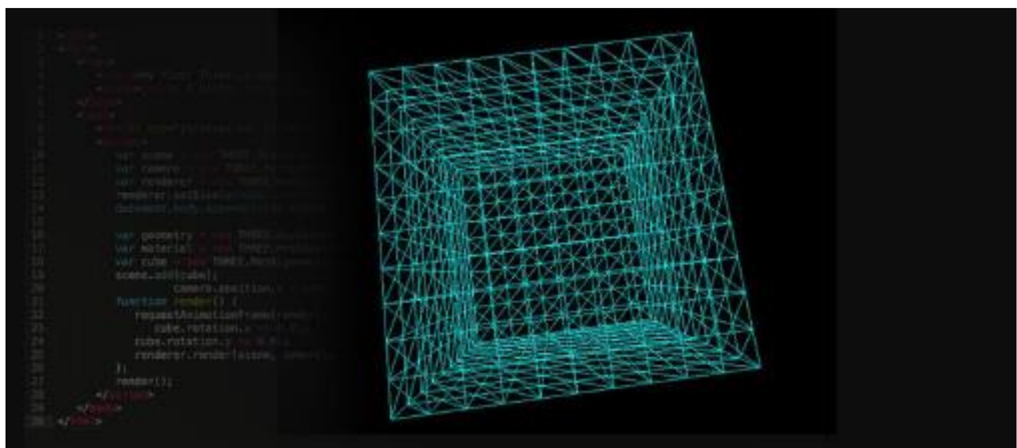
├── Three.js → 3D rendering engine (scene, camera, materials)

├── WebXR API → AR session, hit-test, plane detection

└── TF.js → AI inference (coco-ssd model)

Why pure Three.js (no A-Frame)?

During development it was discovered that A-Frame has its own internal render loop that conflicts with an explicit **immersive-ar** WebXR session. A-Frame starts an **immersive-vr** session (opaque, no passthrough) instead of **immersive-ar**. Switching to pure **Three.js** gave us direct control over both the session type and the render loop.



3.3 Technology Stack

Technology	Version	Role
Three.js	0.157.0	3D rendering, geometry, materials
WebXR Device API	Browser-native	AR session, hit-test, plane detection
TensorFlow.js	4.15.0	In-browser ML inference engine
COCO-SSD	2.2.3	Pre-trained object detection model
getUserMedia API	Browser-native	Camera stream for TF.js analysis

3.4 Application Flow

Page loads

→ TF.js model (coco-ssd) loads asynchronously (~6MB)

→ Check: navigator.xr.isSessionSupported('immersive-ar')

→ "Enter AR" button becomes active

User clicks "Enter AR"

→ Request: navigator.xr.requestSession('immersive-ar', {...})

→ Quest enables Passthrough (camera visible through headset)

→ Simultaneously: getUserMedia() → camera stream for TF.js

→ renderer.setAnimationLoop() starts (XR render loop)

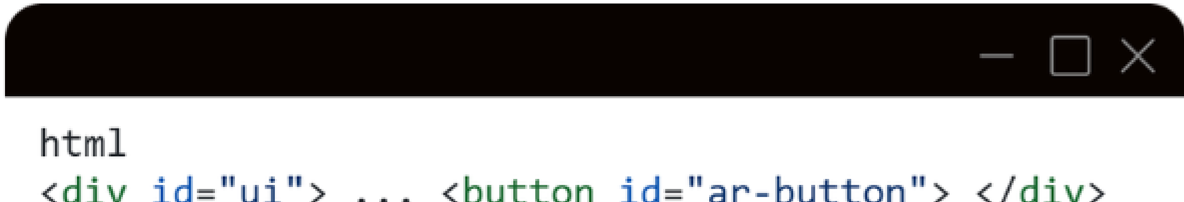
Every frame (~72fps on Quest 3)

- scene.background = null (ensures transparency)
- Hit-test → updates cursor position
- Plane Detection → updates surface visualization
- Every 2s → runDetection() → AI inference
- renderer.render(scene, camera)

3.5 Detailed Code Explanation

1. HTML Structure (lines 1–78)

A hidden <**video**> element receiving the ***getUserMedia***() stream. TensorFlow.js uses it as input for inference — it is never displayed to the user, only analyzed.



An overlay UI layer floating above the **Three.js** canvas. ***pointer-events: none*** on the container but ***pointer-events: all*** on the button only — so the overlay doesn't block 3D interactions.

2. Three.js Initialization (lines 84–109)

```
javascript
const renderer = new THREE.WebGLRenderer({ antialias: true, alpha: true });
renderer.xr.enabled = true;
```

alpha: true creates a WebGL canvas with a transparent alpha channel — this is a prerequisite for passthrough. ***xr.enabled = true*** activates **Three.js**'s built-in WebXR integration.

```
javascript
const camera = new THREE.PerspectiveCamera(70, aspect, 0.01, 20);
scene.add(camera);
```

The camera is **added to the scene**. This is important because objects attached to the camera (child entities) need to be in the scene hierarchy.

```
javascript
const cursorGeo = new THREE.RingGeometry(0.04, 0.06, 32);
cursorGeo.rotateX(-Math.PI / 2);
```

The ring geometry is rotated -90° on the X axis so it lies flat horizontally on detected surfaces. Without this rotation it would stand vertically.

3. makeLabel() — Text Sprite (lines 113–138)

```
javascript
function makeLabel(text, color, bgColor) {
  const canvas = document.createElement('canvas'); // 512x128 px
  // Draws rounded rect + text
  const texture = new THREE.CanvasTexture(canvas);
  const sprite  = new THREE.Sprite(material);
  sprite.scale.set(0.6, 0.15, 1); // 0.6m wide in 3D space
  return sprite;
}
```

THREE.Sprite is an object that always faces the camera (**billboarding**) — ideal for labels in 3D space. It is drawn on an HTML canvas, converted to a ***CanvasTexture***, and applied to a ***SpriteMaterial***.

depthTest: false ensures the label is always visible even if it is geometrically "behind" another 3D object.

4. detectionTo3D() — 2D → 3D Projection (lines 140–163)

This is the mathematical core of the AI-AR integration.

```
javascript
// Normalize bbox center to [-0.5, 0.5]
const nx = (bbox.x + bbox.width/2)  / videoWidth  - 0.5;
const ny = (bbox.y + bbox.height/2) / videoHeight - 0.5;
// Apply camera FOV (70° horizontal)
const fovH = 70 * Math.PI / 180;
const dir  = new THREE.Vector3(
  Math.tan(nx * fovH),
  Math.tan(-ny * fovV), // flip Y (image top-down, WebGL bottom-up)
  -1                    // forward in camera space (Three.js uses -Z)
).normalize();
// Rotate into world space using the current XR camera orientation
dir.applyQuaternion(camera.quaternion);
// World position = camera position + direction × distance
return camera.position.clone().addScaledVector(dir, placeDist);
```

Example: If the AI detects a chair in the left third of the video, ***nx ≈ -0.17****. This converts to a negative angle (left of the gaze axis). The label is placed 1.8m in front of the camera in that direction.*

5. makeBox3D() and bbox2dSize3D() — 3D Bounding Boxes (lines 165–182)

```
javascript
function makeBox3D(w, h, colorHex) {
  const edges = new THREE.EdgesGeometry(new THREE.BoxGeometry(w, h, 0.01));
  return new THREE.LineSegments(edges, material);
}
```

EdgesGeometry + LineSegments *renders only the edges of a box — the effect of a thin wireframe border with no filled surface.*

```javascript
function bbox2dSize3D(bboxW, bboxH, videoW, videoH, dist) {
  const fovH   = 70 * Math.PI / 180;
  const totalW = 2 * dist * Math.tan(fovH / 2); // total visible width at given dist
  return {
    w: (bboxW / videoW) * totalW,  // bbox fraction → meters
    h: (bboxH / videoH) * totalH
  };
}
```

*The **totalW** formula is standard perspective projection: **at distance d**, the visible width depends on the FOV. From this we derive how many meters correspond to the pixels of the bounding box.*

6. runDetection() — AI Inference Pipeline (lines 205–247)

```javascript
async function runDetection() {
  const predictions = await cocoModel.detect(tfVideo);
  // Clear old labels and boxes
  activeLabels.forEach(({ sprite, box }) => { scene.remove(sprite); scene.remove(box); });
  activeLabels = [];
  predictions.forEach(pred => {
    if (pred.score < 0.5) return; // Confidence threshold: 50%
    // ...create sprite + box...
    activeLabels.push({ sprite, box, expireAt: Date.now() + 4000 });
  });
}
```

***cocoModel.detect(tfVideo)** accepts a <video> element directly — TF.js internally reads pixel data from the current video frame.*

***pred.bbox** format: **[x, y, width, height]** in pixels.*

*Each detection creates a **JS(sprite, box)** pair that lives for 4 seconds.*

7. WebXR Session — Key Details (lines 274–285)

```javascript
const session = await navigator.xr.requestSession('immersive-ar', {
  requiredFeatures: ['local-floor'],
  optionalFeatures: ['hit-test', 'plane-detection']
});
renderer.xr.setReferenceSpaceType('local-floor');
await renderer.xr.setSession(session);
```

Why **immersive-ar** and not **immersive-vr**?

- **immersive-vr** = opaque black background (VR helmet experience)
- **immersive-ar** = passthrough camera + virtual content overlaid on top

local-floor reference space fixes the coordinate system to the room floor — **y=0** is at floor level.

hit-test and **plane-detection** are optional because they are not supported on all devices and browser versions — declaring them as optional prevents the session from failing if they're unavailable

8. Render Loop (lines 335–395)

```javascript
renderer.setAnimationLoop(function(time, frame) {
  scene.background = null;             // Ensures transparency every frame
  renderer.setClearColor(0x000000, 0); // Alpha = 0 (fully transparent)
  // ...hit-test, plane detection, AI...
  renderer.render(scene, camera);
});
```

Why **setAnimationLoop** and not **requestAnimationFrame**?

The **Three.js** XR render loop must be integrated with the WebXR frame callback. **renderer.setAnimationLoop()** automatically binds to the XR session when **renderer.xr.enabled = true**. The alternative (manually calling **session.requestAnimationFrame()**) requires calling **renderer.render()** manually but risks missing the correct XR view matrices that **Three.js** applies internally.

scene.background = null must be called **every frame** because Three.js may reset this value internally during certain operations.

9. Plane Detection (lines 355–378)

```
javascript
session.detectedPlanes.forEach(plane => {
  if (!detectedPlanes.has(plane)) {
    // New surface detected — create a mesh
    const label = plane.semanticLabel; // 'floor', 'wall', 'table'...
    const mesh = new THREE.Mesh(PlaneGeometry, transparentMaterial);
    scene.add(mesh);
    detectedPlanes.set(plane, mesh); // store reference
  }
  // Every frame, update the surface pose
  const pose = frame.getPose(plane.planeSpace, xrRefSpace);
  mesh.position.copy(pose.transform.position);
  mesh.quaternion.copy(pose.transform.orientation);
});
```

plane.planeSpace is an XRSpace that tracks the physical surface. **frame.getPose()** returns its pose in the chosen reference space.

The **detectedPlanes** map **(Map<XRPlane, THREE.Mesh>)** prevents creating duplicate meshes for the same surface across frames.

3.5 Known Limitations

Limitation	Reason
AI labels are not pixel-perfect aligned with object	The Quest passthrough cameras and getUserMedia deliver different streams with different FOV and optical calibration
Plane detection requires Quest Room Setup to be completed	The headset must have previously scanned the room
getUserMedia may be denied on some devices	Depends on browser permissions
AI inference is not real-time (runs every 2s)	coco-ssd is a relatively heavy model; lighter alternatives (MobileNet SSD) would give higher throughput

3.5 Source code

https://github.com/bciric1/ARVR

http://vr.aiforvet.eu/