TEAM PARADUCKS

FUTURE ENGINEERS

TABLE OF CONTENT

WHO ARE WE?

Team Name: Paraducks

Category: Future Engineers

School: Bombay International School

Grade: 10

MEMBERS

Neel

Dev

Tara

FUTURE ENGINEERS

MOBILITY MANAGEMENT

FUTURE ENGINEERS

MOBILITY MANAGEMENT

MOTORS

We are following the motor limit and are using one Servo motor and one DC motor

SERVO MOTOR

We used a servo to turn the front axle, controlling robot steering, while the wheels remained uncontrolled to move with the rear axle. The robot dimensions allow a **maximum turning range of 90°**. The axle is a disjoint shaft to prevent the wheels from slipping.

Digital servo: Adequate torque for robot weight and accurate, fast turning, space and power efficient.

DC MOTOR

We connected our DC motor to a bevel gear to power the entire rear axle with the singular motor, **powered by the motor driver** in turn directly sourcing power from the buck converter which is connected to the battery

12V brushed DC motor: Affordable, simple, reliable, power efficient, consistent torque and speed, adequate to move entire robot at moderate speeds

Differential gearbox: Allows wheels to rotate at different speeds, prevents slippage and allows a minimum of 10 cm turning radius which allows us to parallel park

Cytron MD10C R3 Motor Driver: High continuous current supply.

MOBILITY MANAGEMENT

ENGINEERING FACTOR

The design needed to be **agile** and have a **very small radius of turn.** This was achieved by using a smaller wheelbase and a stacked CNC made Polycarbonate sheets design to accommodate displaced electronic components.

A metal differential is used to couple the rear disjoint shaft together, for better power delivery to both the rear wheels. This leads to **close to no slippage while turning**.

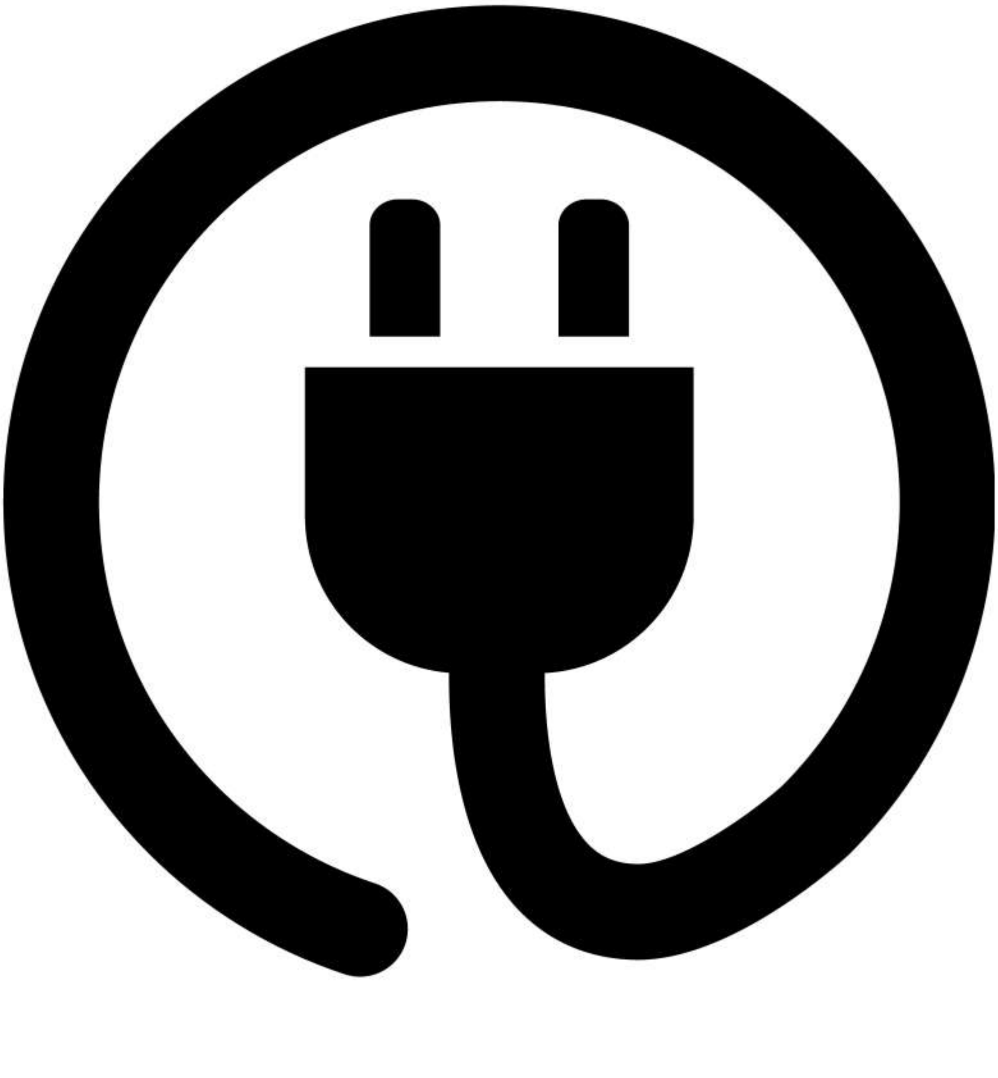
The motor is mounted vertically with the shaft **facing down**. The stability of the bot was another priority, taken care by **very low Centre of gravity** due to the battery being placed below the deck.

The camera is mounted 11" of the ground with a 20° incline which gives us a **very large POV** for detection of blocks.

The C1 Lidar placed in front helped to get a better view of the block surrounding at all times, which allows for quick turning around blocks. The lidar also acts as an dditional sensor for circumventing around the blocks. This paired with the TF-minis we have on the left, right, and front ensures we are able to manipulate the servo to turn **efficiently** every time.

FUTURE ENGINEERS

POWER AND SENSE MANAGEMENT



FUTURE ENGINEERS

COMPONENTS


RPLidar C1

Hikvision Camera

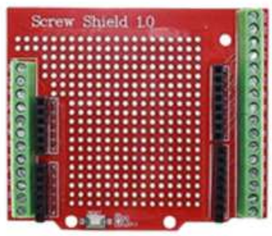
Google Coral


TFmini LiDAR

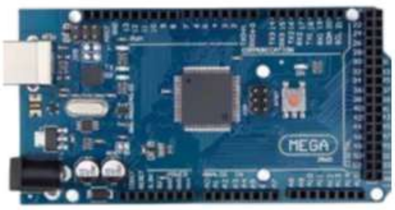

RasPi 4B


REES52 12V-5V Converter

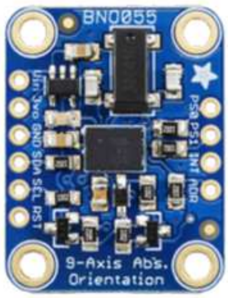

Switch



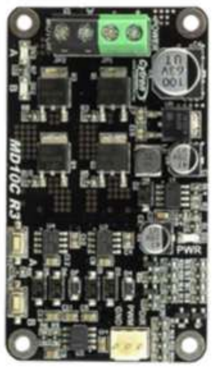
Screw Shield



Arduino Mega 2560



BNO085 IMU



MD10C Motor Driver

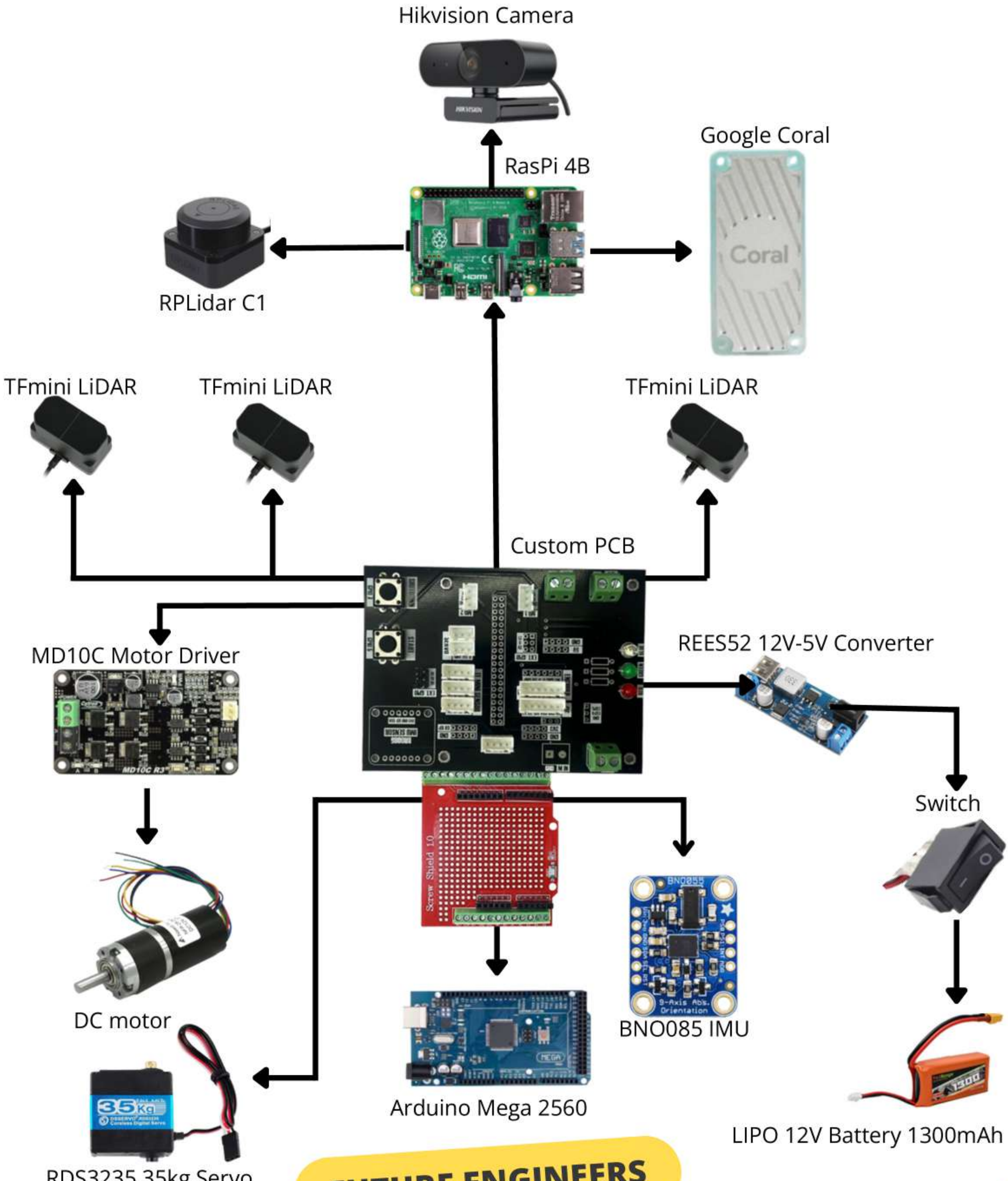
RDS3235 35kg Servo


DC Motor


Custom PCB

LIPO 12V Battery 1300mAh

CIRCUIT



FUTURE ENGINEERS

POWER

Power Supply: Pro-Range LiPo 11.1 V 1300mah 3 cell battery

The motor driver is **directly** connected and powered by the battery, which in turn powers the motor.

We use a **12V -> 5V buck converter module** to step down voltage which is distributed by our custom PCB to all sensors and the following methods for the remaining components:

Raspberry Pi 4B + Coral :
Connected and powered by the USB-A port in the buck converter to its USB-C power port.
Coral increases FPS for the model as its an external GPU

Arduino MEGA:
Connected and powered from the RPi's USB-A port to its USB-B power port. Same connection used for serial communication between the two.

MICROCONTROLLERS

We initially began prototyping with an Arduino UNO, but found many flaws in it, primarily its image processing with a HuskyLens.

We now use combination of :

Raspberry Pi 4B + Coral: Primary microprocessor, where programs are stored and programmed in Python. Far better object detection and multiprocessing due it its multiple cores. Doesn't give enough FPS. Added Google coral which is an external GPU to boost model processing speeds which leads to an increased FPS.

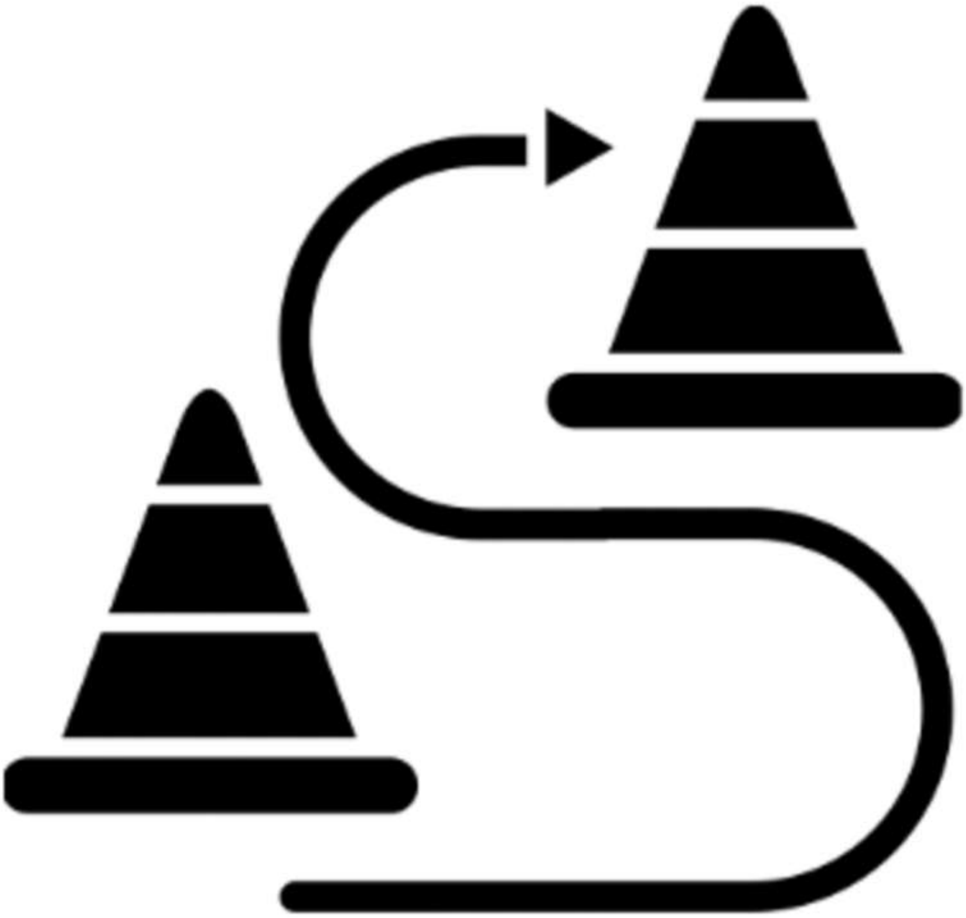
Arduino MEGA 2560 Rev3: Secondary microcontroller. Used for its superior compatibility and libraries for serial communication with sensors: Motor encoder and IMU. It communicates with raspberry pi via UART communication.

Note: We also used an ESP32 DevKit for a while, but compatibility and connectivity issues forced us to replace it with the Arduino MEGA.

SENSORS

Name	Description	Use
C1 LiDAR	Compact laser sensor, measures distance with high precision, wide field of view	Mapping surroundings and obstacle avoidance
Hikvision Camera	High-resolution color camera, supports video recording and streaming	Object detection, visual navigation
TF-mini LiDAR	Small and cost-effective LiDAR, up to 12 meters range	Detecting nearby objects, simple distance sensing
Motor Encoder	Rotary encoder, counts wheel or shaft rotations	Monitoring robot movement and controlling speed
IMU	Contains gyroscope and accelerometer for motion tracking	Tracking orientation, measuring acceleration

OBSTACLE MANAGEMENT



AI DETECTION

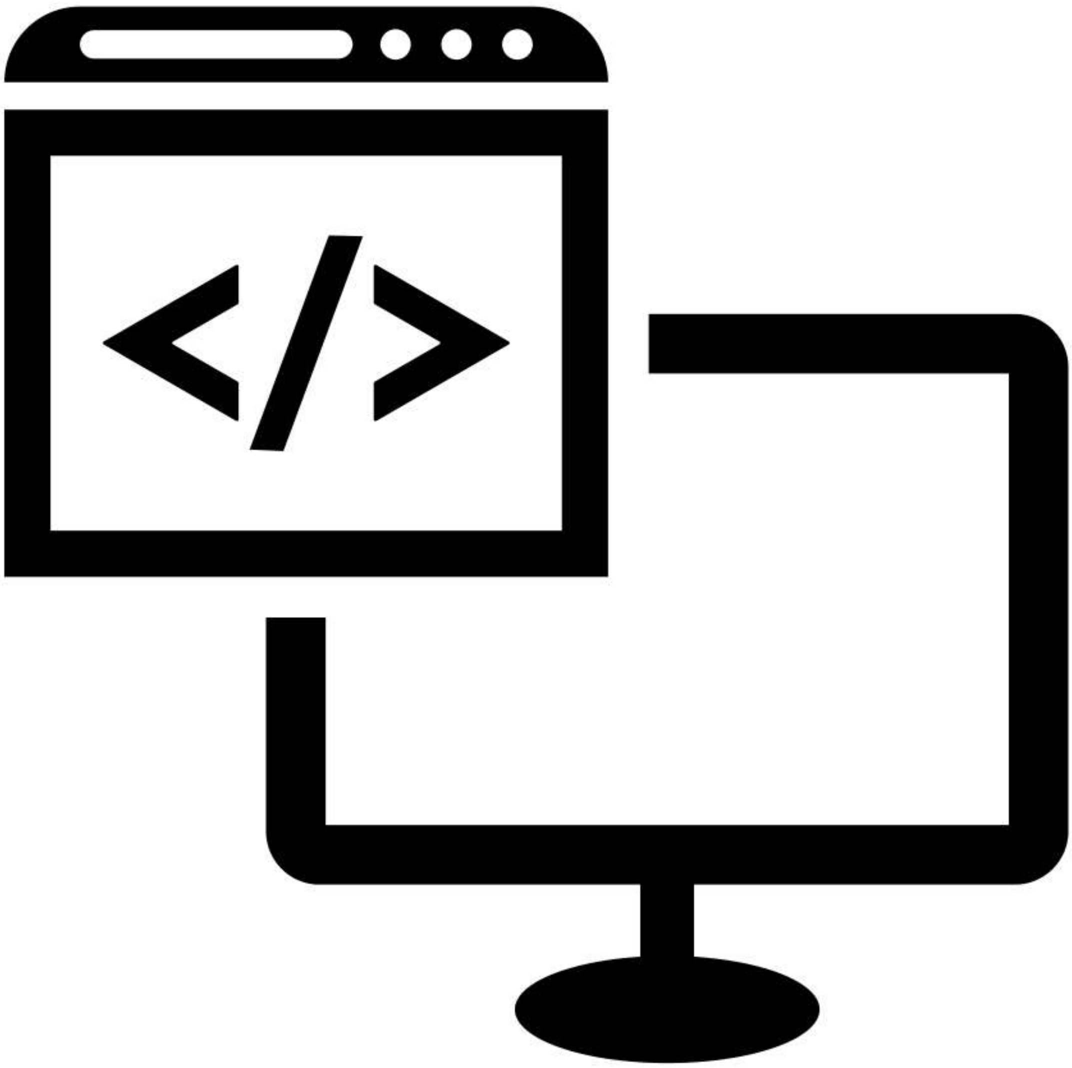
AI Model

MobileNet SSD v2 is an efficient, lightweight deep learning model designed for real-time object detection on mobile and embedded devices. It combines the MobileNet V2 architecture, which extracts important features from images using advanced convolution techniques,

Why we are using this:

- **Efficiency on Low-Power Devices:** Mobile-Net SSD v2 is highly optimized for mobile and embedded hardware. It can deliver accurate object detection at fast speeds even on resource-constrained devices like Raspberry Pi and microcontrollers—something traditional models or manual color sensors cannot do reliably.
- **Real-Time Detection and Versatility**: This model detects blocks and classifies their features in a single pass, giving rapid, real-time feedback that's ideal for robotics tasks. It is robust against varied lighting, camera angles, and backgrounds compared to simpler methods like color sensors or thresholding, which often require manual calibration.
- **Easy Customization and Wider AI Support**: Mobile-Net SSD v2 is compatible with popular AI frameworks and can be custom-trained for your specific block colors and environments, unlike traditional threshold-based pipelines or hardware color sensors that are harder to adapt.

PROGRAMMING



FUTURE ENGINEERS

PROGRAMMING

Key Functions:

TFMini Data: Calculates checksum of 9-bit data packages sent by TFMini, accepts value if correct and converts to cm.

Set Angle: Uses PWM (pulse-width modulation) to set angle, converts degree input to pulse-width

Button Toggle: Compares previous and current state of the button (on/off) to see if it is pressed to start or stop the robot.

Running Encoder: Establishes UART communication with Arduino MEGA; reads, decodes and strips lines of data received

PID controller: Used to decrease error between desired and servo position

Programming Language:

Python

Turning Logic

The turning logic is based on LIDAR sensor readings and special direction flags:
If there is an obstacle close in front (lidar_front<900lidar_front<900), and space to the right (lidar_right>1500lidar_right>1500), and the right flag is set (right_f.value) but not the left flag, the robot triggers a right turn (turn_trigger.value = True).
If there is an obstacle close in front, and space to the left (lidar_left>1500lidar_left>1500), and the left flag is set (left_f.value) but not the right flag, the robot triggers a left turn.
If neither condition is met, the robot does not trigger a turn (turn_trigger.value = False).

FUTURE ENGINEERS

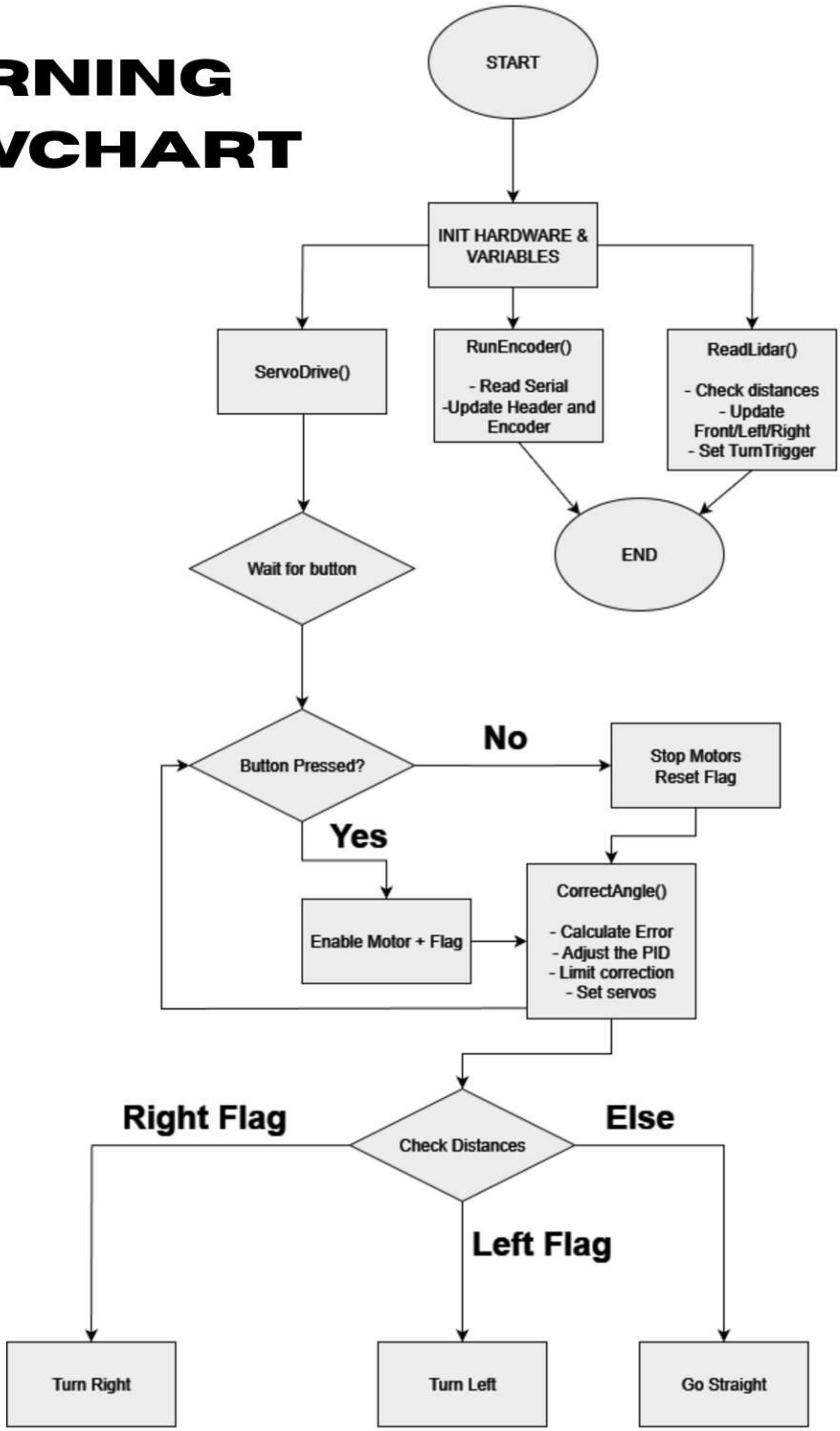
PSEUDOCODE

TURN

```
IF sp_angle > 180 THEN
 sp_angle ← sp_angle - 360
ENDIF
IF previous_angle ≠ angle THEN
 IF prev_sp ≠ sp_angle THEN
 sp_angle ← 360 - sp_angle
 ENDIF
 prev_sp ← sp_angle

 WHILE (angle - previous_angle) > 1
DO
 lidar_angle ← (previous_angle + 1)
MOD 360
 lidar_distance ← previous_distance
 previous_angle ← lidar_angle

  IF lidar_angle = ((0 + imu_r +
sp_angle) MOD 360) THEN
 lidar_front ← lidar_distance
 lidar_f ← lidar_front
 ENDIF

IF lidar_angle = ((90 + imu_r +
sp_angle) MOD 360) THEN
 lidar_left ← lidar_distance
 ENDIF

 IF lidar_angle = ((270 + imu_r + sp_angle)
MOD 360) THEN
 lidar_right ← lidar_distance
 ENDIF

 # Turning decision based on lidar readings
and flags
 IF (lidar_front < 900 AND lidar_right > 1500)
AND right_flag = TRUE AND left_flag = FALSE
THEN
 turn_trigger ← TRUE
 ELSE IF (lidar_front < 900 AND lidar_left >
1500) AND left_flag = TRUE AND right_flag =
FALSE THEN
 turn_trigger ← TRUE
 ELSE
 turn_trigger ← FALSE
 ENDIF

 # Print current status (for debugging)
 PRINT "turn_trigger: ", turn_trigger, "
lidar_front: ", lidar_front, ", lidar_left: ",
lidar_left, ", lidar_right: ", lidar_right, "
sp_angle: ", sp_angle, " head: ", head
 ENDWHILE
ENDIF
```

FUTURE ENGINEERS

TURNING FLOWCHART



PSEUDOCODE

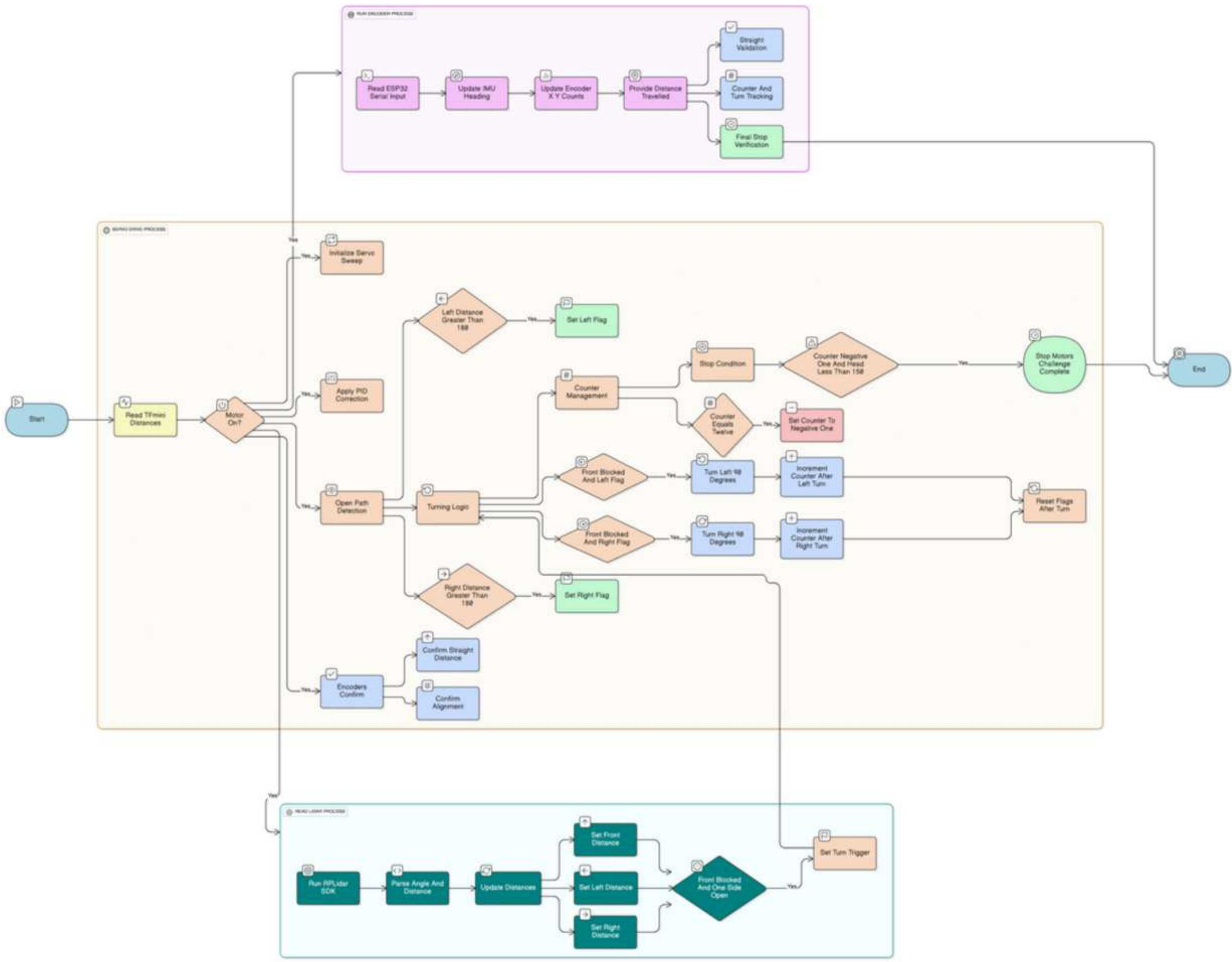
OPEN CHALLENGE

```
BEGIN PROGRAM

 INIT hardware, reset Arduino, open serial, set variables

 PROCEDURE CorrectAngle(Target, Heading, L, R)
  error ← Heading - Target
  IF error > 180 THEN
   error ← error - 360
  ENDIF
  correction ← kp*error + kd*(error-prevError) + ki*
(totalError+error)

  IF L < 15 THEN
   correction ← correction - 20
  ELSE IF
   R < 15 THEN correction ← correction + 20
  ENDIF

  LIMIT correction TO -30..30
  SetServo(90 - correction)
 ENDPROCEDURE

 PROCEDURE ServoDrive()
  WAIT button
  LOOP
   READ front, left, right
   IF button pressed THEN
    ENABLE motor
    IF no flags THEN
     IF right > 180 THEN
      RightFlag ← TRUE
     ELSE IF left > 180 THEN
      LeftFlag ← TRUE
     ENDIF
    ENDIF
   CALL CorrectAngle(Target, Heading, left, right)
   IF RightFlag AND right > 130 AND front < 75 AND NOT
Trigger THEN
    counter ← counter+1
    Target ← (90*counter) MOD 360
    Trigger ← TRUE
   ELSE IF LeftFlag AND left > 130 AND front < 75 AND NOT
   Trigger THEN
    counter ← counter+1
    Target ← -(90*counter) MOD 360
    Trigger ← TRUE
   ELSE IF (right < 85 OR left < 85) AND front > 75
THEN
    Trigger ← FALSE
   ENDIF
  ELSE
   STOP motor, RESET flags
   CALL CorrectAngle(0, Heading, left, right)
  ENDIF
 ENDLOOP
ENDPROCEDURE

PROCEDURE RunEncoder()
 LOOP READ serial → update Heading,
EncoderCounts
 ENDLOOP
ENDPROCEDURE

PROCEDURE ReadLidar()
 LOOP
  READ angle, dist → update Front, Left, Right
  IF Front < 900 AND (Right > 1500 OR Left > 1500)
THEN
   TurnTrigger ← TRUE
  ELSE
   TurnTrigger ← FALSE
  ENDIF
 ENDLOOP
ENDPROCEDURE

RUN ServoDrive, RunEncoder, ReadLidar IN
PARALLEL

END PROGRAM
```

FUTURE ENGINEERS

FLOWCHART

OPEN CHALLENGE



FUTURE ENGINEERS

PSEUDOCODE

OBSTACLE CHALLENGE

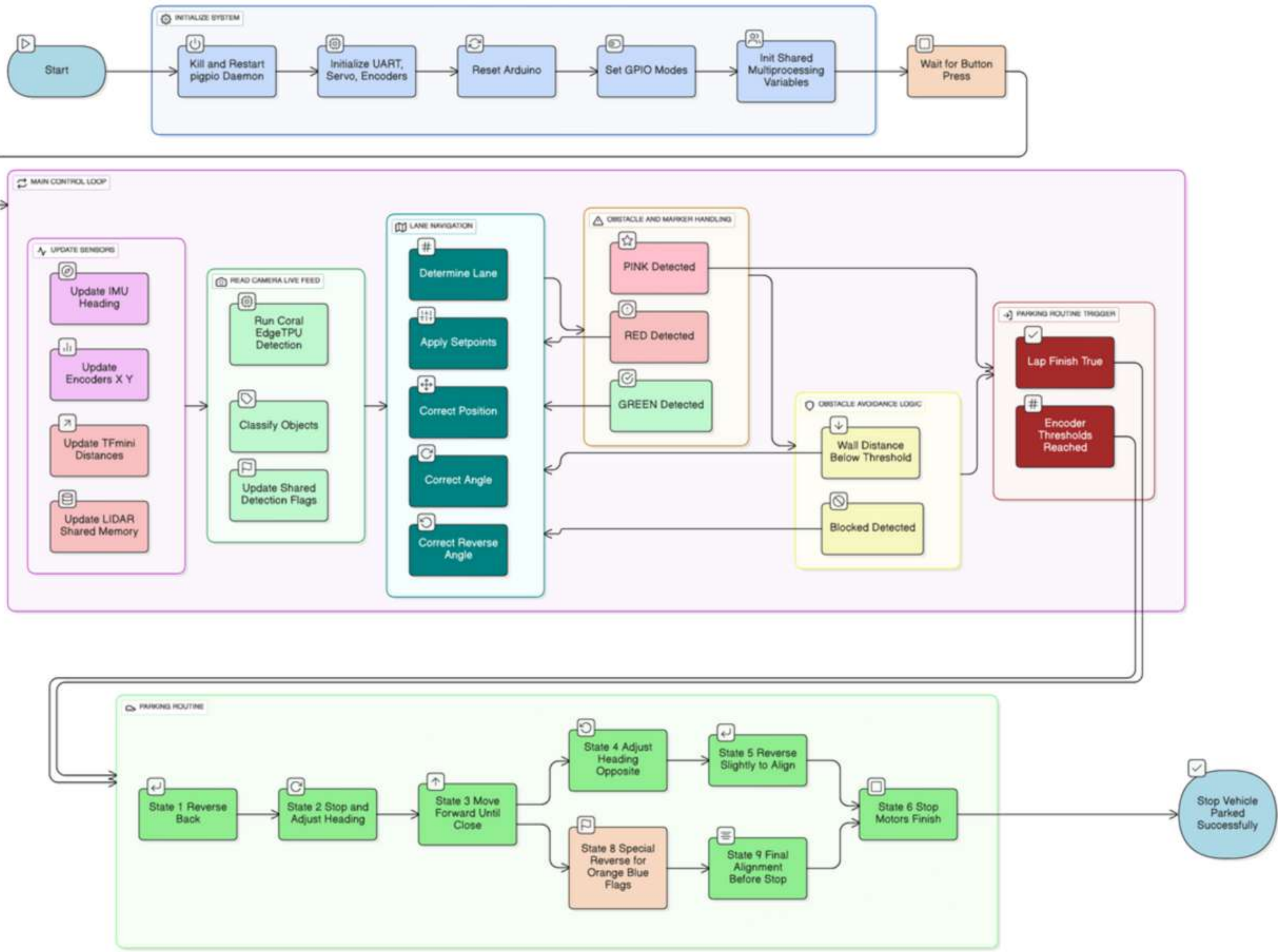
```
PROCEDURE Main
   INITIALISE all pins and devices
   RESET Arduino
   CREATE Servo using servo_pin
   CREATE TFmini using RX_Head, RX_Left, RX_Right, RX_Back
   CREATE EncoderCounter
   inParkingAtStart ← FALSE
   lap_finish ← FALSE
   continue_parking ← FALSE
   blue_flag ← FALSE
   orange_flag ← FALSE
   parking_flag ← FALSE
   counter ← 0
   WHILE TRUE
      IF ButtonIsPressed() THEN
         DriveRoutine()
      ELSE
         StopMotors()
         heading_angle ← 0
         counter ← 0
         CALL CorrectAngle(heading_angle, CurrentHeadValue())
      ENDIF
   ENDWHILE
ENDPROCEDURE
PROCEDURE DriveRoutine
   WHILE ButtonIsPressed()
      CALL UpdateSensors()  'Get all necessary sensor data

      IF lap_finish = FALSE THEN
         IF ParkingDetected() THEN
            inParkingAtStart ← TRUE
            IF TF_Left < 20 AND TF_Head < 200 THEN
               RightParking()
            ELSEIF TF_Right < 20 AND TF_Head < 200 THEN
               LeftParking()
            ENDIF
         ENDIF
         CALL DriveLogic()
      ELSE
         CALL ParkingLogic()
      ENDIF
   ENDWHILE
ENDPROCEDURE
```

```
PROCEDURE CorrectAngle(setPoint_gyro, heading)
   error_gyro ← heading - setPoint_gyro
   IF error_gyro > 180 THEN
      error_gyro ← error_gyro - 360
   ENDIF
   correction ← kp * error_gyro + kd * (error_gyro - prevErrorGyro)
+ ki * totalErrorGyro
   IF correction > 30 THEN
      correction ← 30
   ELSEIF correction < -30 THEN
      correction ← -30
   ENDIF
   prevErrorGyro ← error_gyro
   SET_SERVO_ANGLE(90 - correction)
ENDPROCEDURE

PROCEDURE UpdateMotors(power)
   SET_PWM_DUTYCYCLE(pwm_pin, power)
   SET_DIRECTION(direction_pin, 1)
ENDPROCEDURE

PROCEDURE StopMotors()
   SET_PWM_DUTYCYCLE(pwm_pin, 0)
   SET_DIRECTION(direction_pin, 0)
ENDPROCEDURE
PROCEDURE DriveLogic
   IF DetectGreenBlock() THEN
      power ← 70
      CALL CorrectPosition(setPointL, heading_angle, x, y, counter, blue_flag,
orange_flag, ...)
   ELSEIF DetectRedBlock() THEN
      power ← 70
      CALL CorrectPosition(setPointR, heading_angle, x, y, counter, blue_flag,
orange_flag, ...)
   ELSEIF DetectPinkBlock() THEN
      IF blue_flag THEN
         IF TF_Left < 30 THEN
            CALL CorrectAngle(heading_angle + 10, CurrentHeadValue())
         ELSE
            CALL CorrectAngle(heading_angle, CurrentHeadValue())
         ENDIF
      ELSEIF orange_flag THEN
         IF TF_Right < 30 THEN
            CALL CorrectAngle(heading_angle - 10, CurrentHeadValue())
         ELSE
            CALL CorrectAngle(heading_angle, CurrentHeadValue())
         ENDIF
      ENDIF
   ELSE
      power ← 85
      CALL CorrectPosition(setPointC, heading_angle, x, y, counter, blue_flag,
orange_flag, ...)
   ENDIF
   CALL UpdateMotors(power)
ENDPROCEDURE
```

FUTURE ENGINEERS

FLOWCHART

OBSTACLE CHALLENGE



OBJECT DETECTION MODEL

Create a Roboflow dataset with 5193 annotated and augmented images.

Download the dataset as a ZIP file in TensorFlow TFRecord format.

Unzip and extract test, train, and validation images on a Linux system.

Install Ultralytics and set up a virtual environment compatible with EdgeTPU.

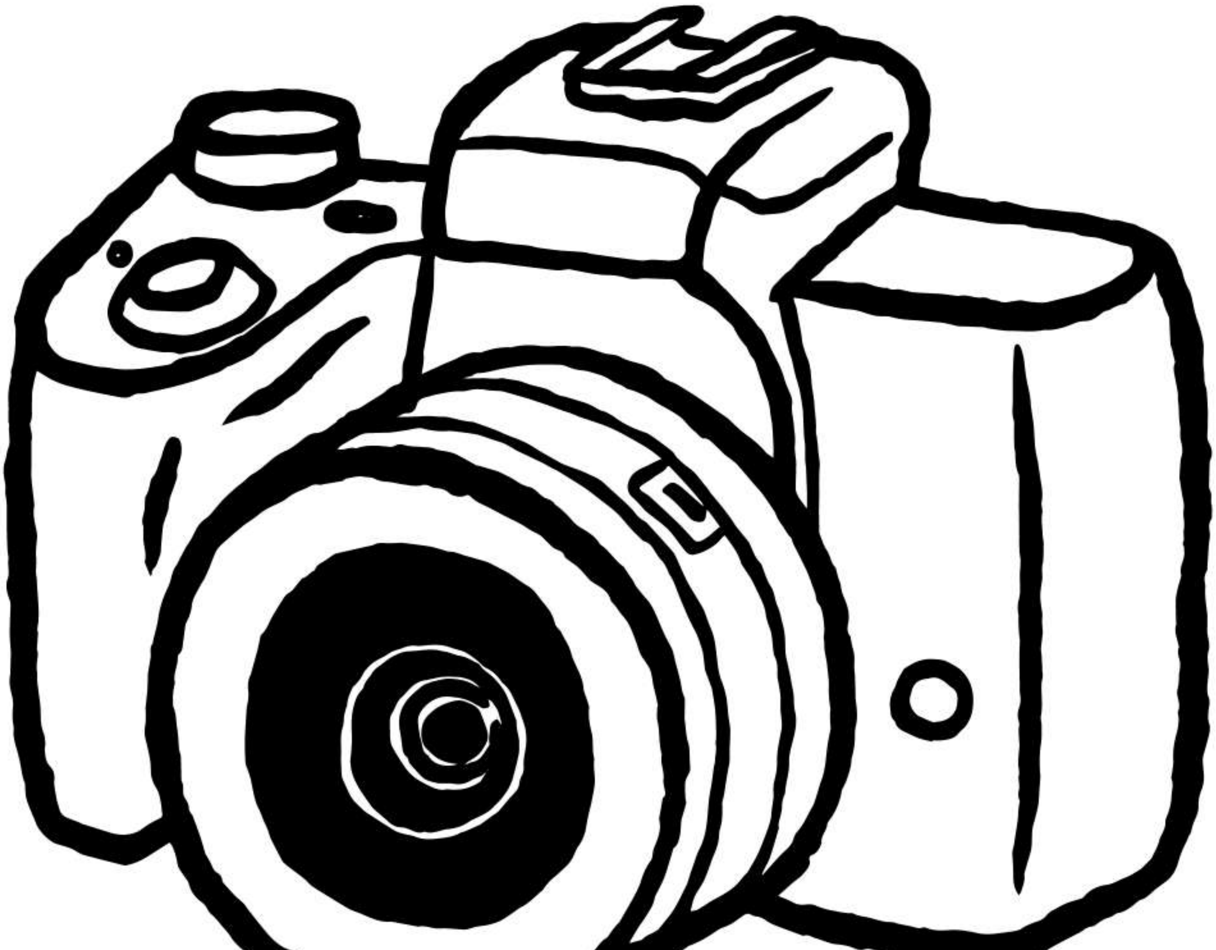
Start model training after confirming that there are no import errors.

After training, compile the MobileNet SSD v2 model file using the EdgeTPU compiler with all quantisations (full integer, float-32, etc.).

Import the compiled, quantised model file into the code directory.

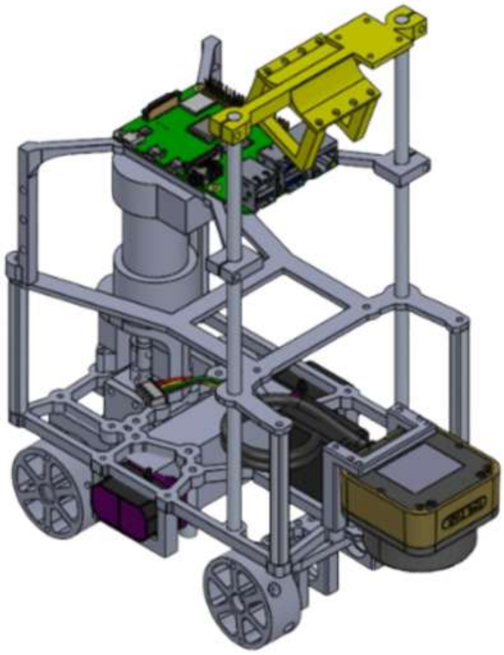
Copy the model path and test performance by printing FPS with both CPU and Google Coral; also, check the bounding box outputs.

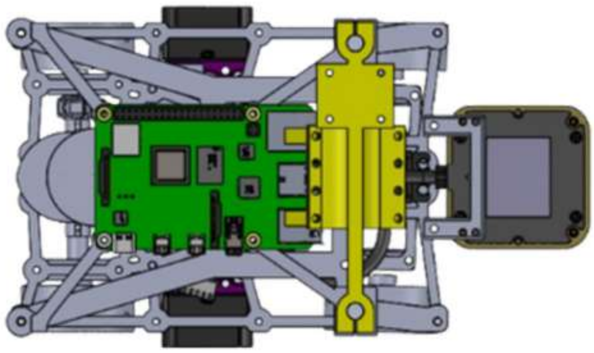
GALLERY

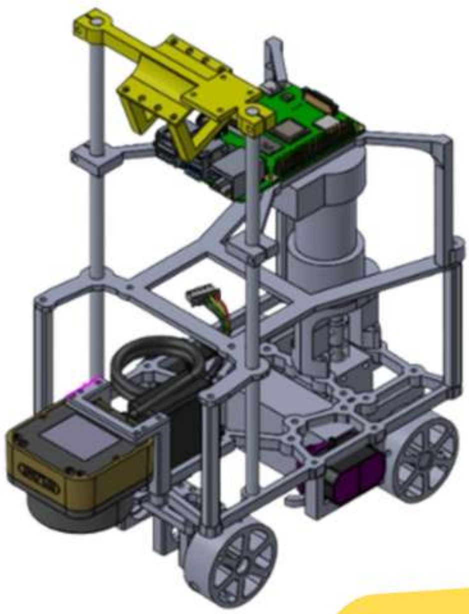


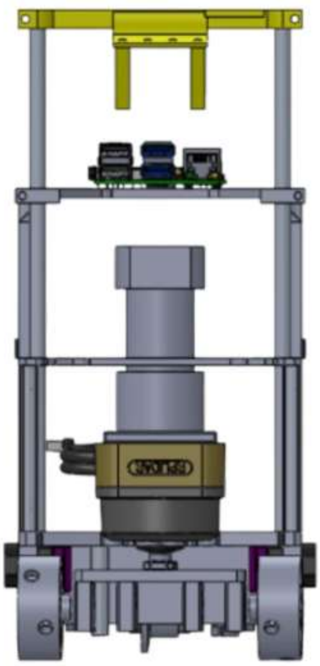
CAD IMAGE

We designed our robot's online model using Solidworks as our software for computer aided design (CAD) which helped in a very high accuracy for sizing of components and creation of custom 3D printed mounts, helping us stay within the stipulated robot limitations.

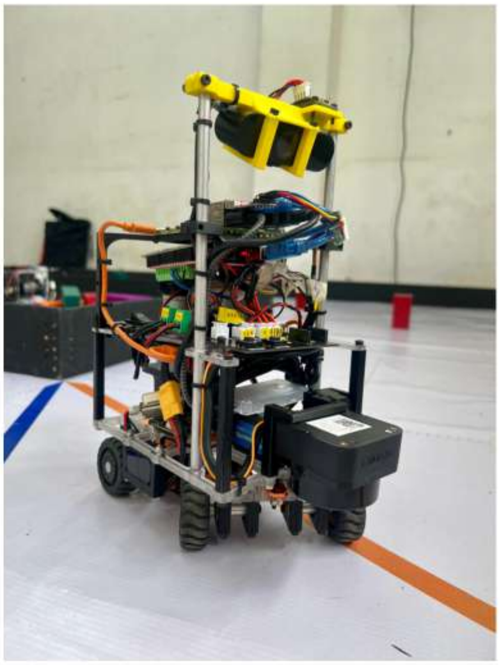


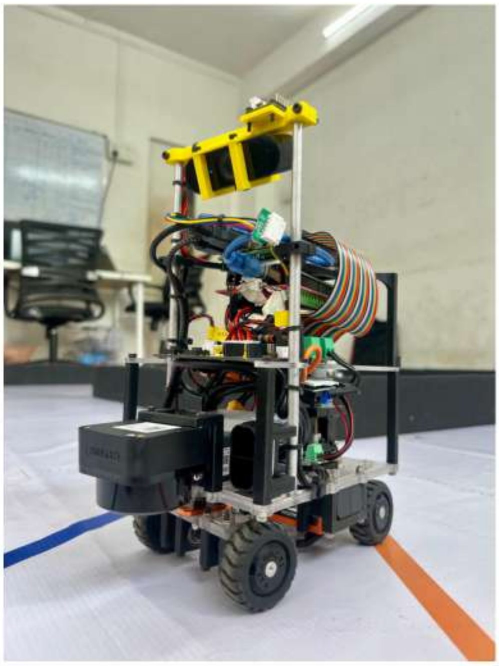


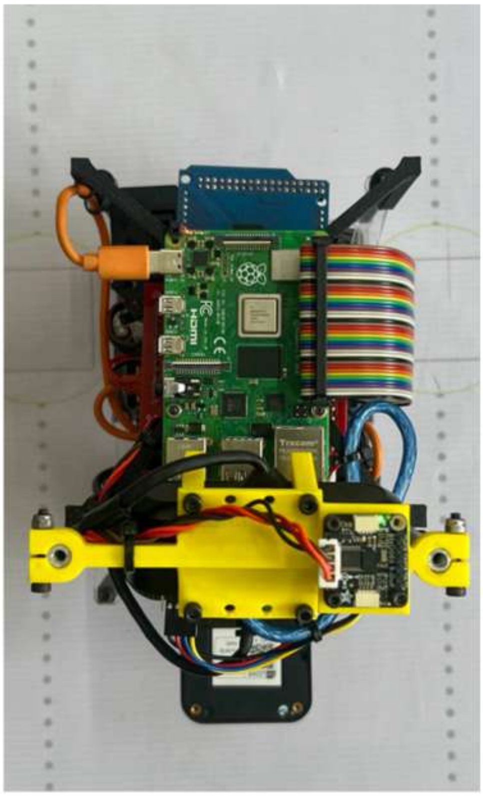




IRL IMAGES







FUTURE ENGINEERS

YOUTUBE CHANNEL

FUTURE ENGINEERS

GITHUB

FUTURE ENGINEERS