Future Engineering

Engineering Journal

Team name: SSTEVE

Team ID: FE25-314-02

Team Members: Joab, Gavin, James

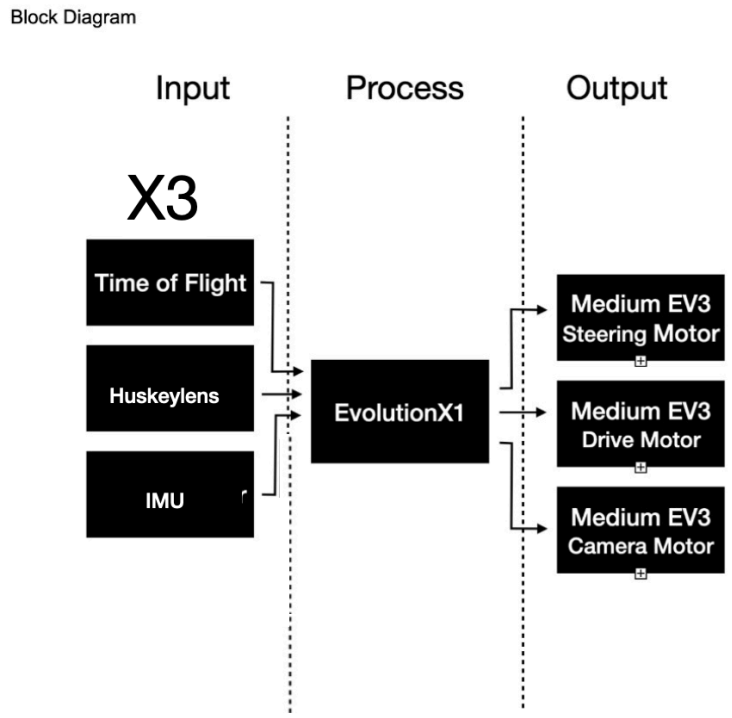
Contents

Hardware Overview

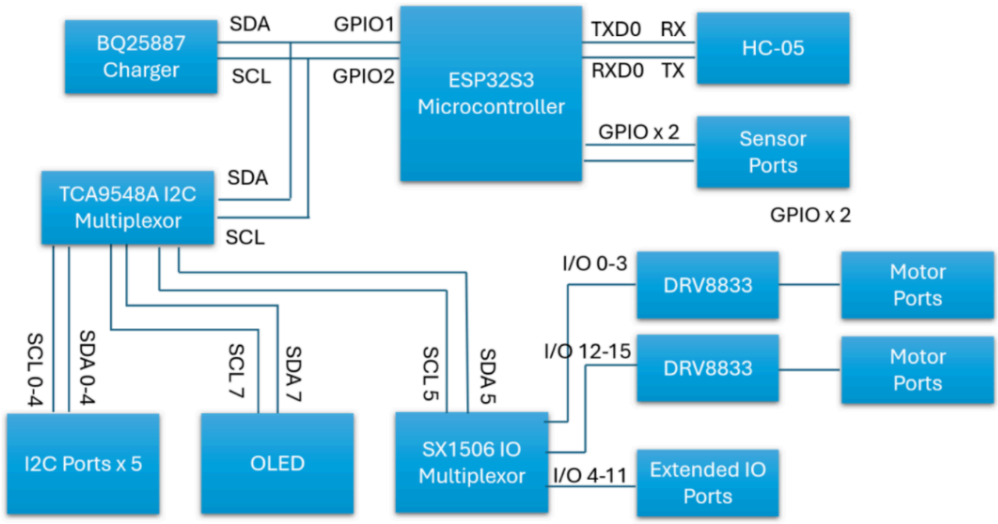
Table of Components Used:

Description	Model	Quantity
EV3 Motor	Medium motor	2
Time of Flight	VL53L1X	3
IMU	BNO055	1
Camera	HuskyLens	1
Microcontroller	Evolution X1 (esp32 base)	1

Block Diagram of input/output:



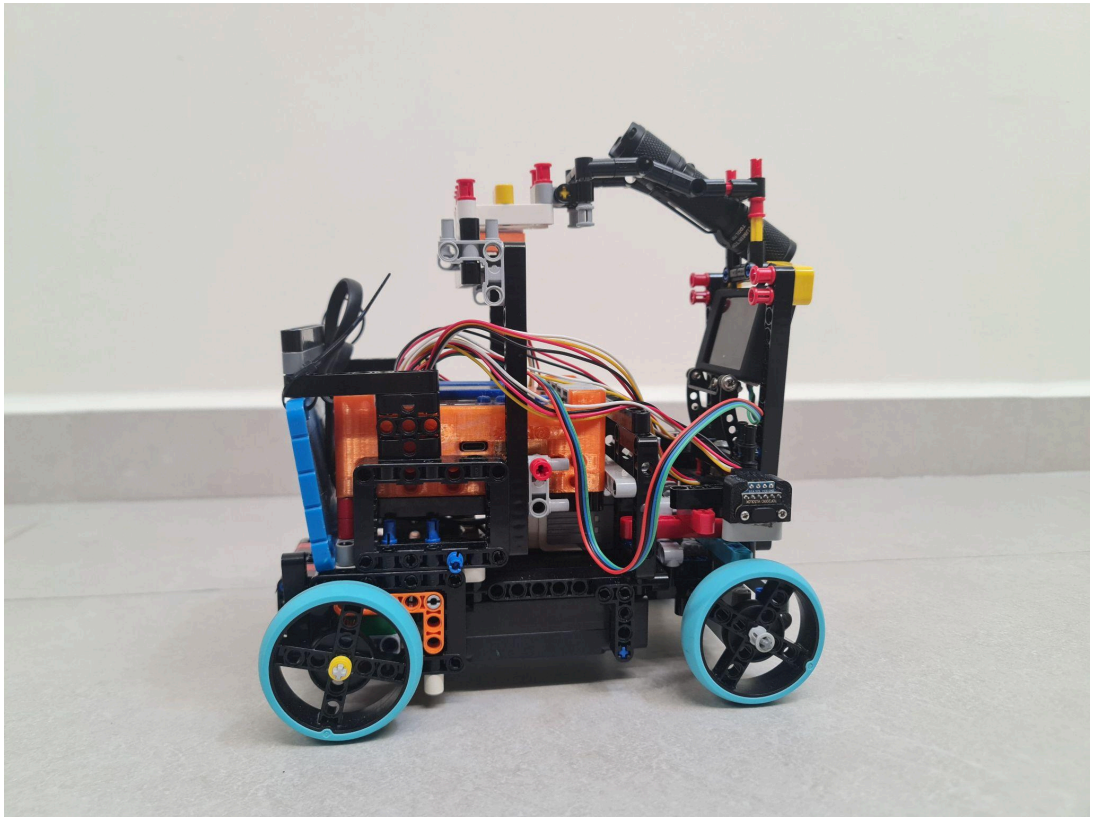
Microcontroller system diagram:



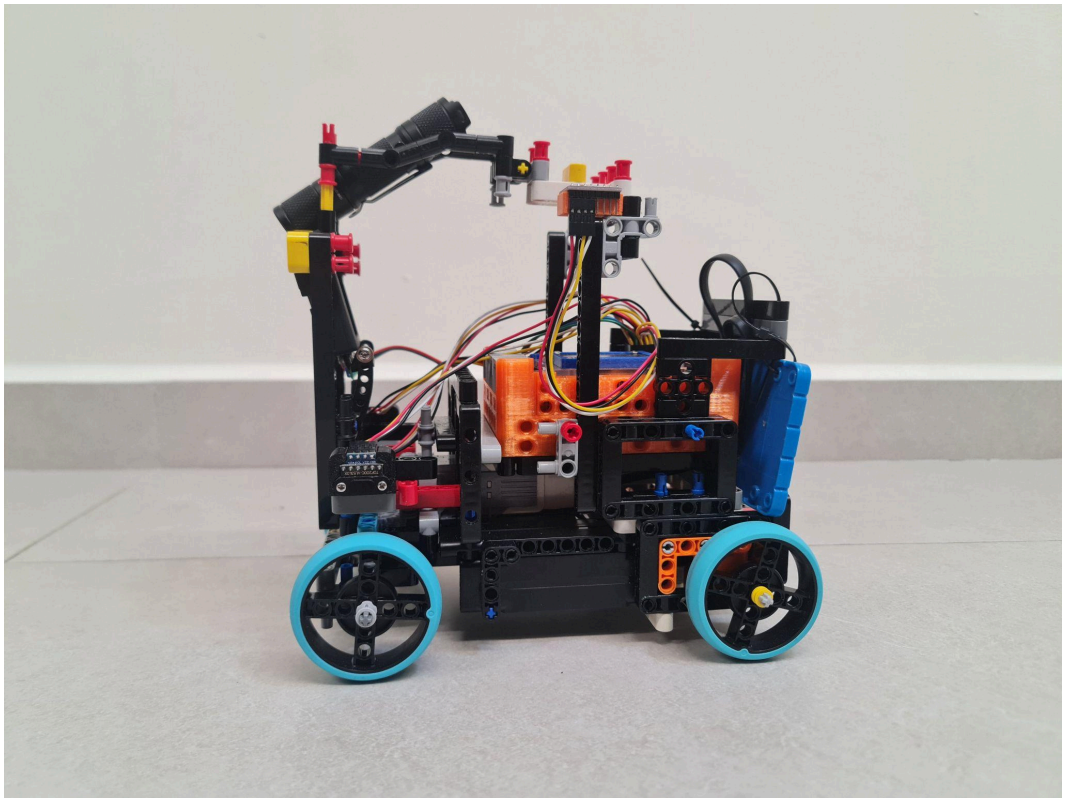
Chassis

Our chassis was constructed entirely from Lego parts. We selected Lego as our material of choice due to its lightweight nature, modularity, and ease of use. The abundance of Lego pieces readily available to us made it a practical and cost-effective option. Its intuitive snap-fit system and wide variety of structural components enabled us to build a highly customisable and adaptable chassis. This flexibility proved especially useful during prototyping, as we could make quick design changes without needing specialised tools or equipment. The simplicity of the system also allowed for rapid assembly and disassembly, which greatly supported our iterative design process.

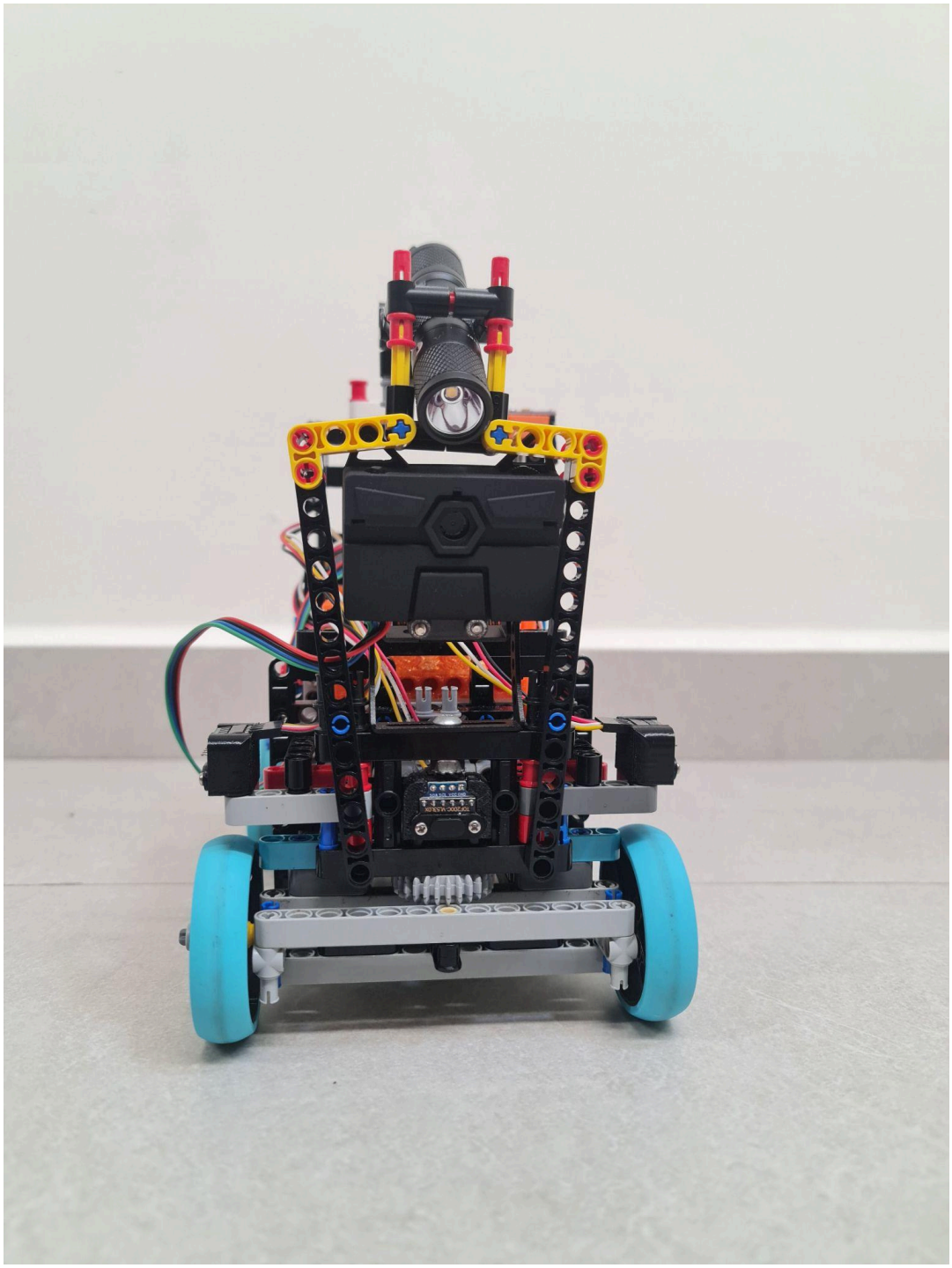
Right View



Left View:

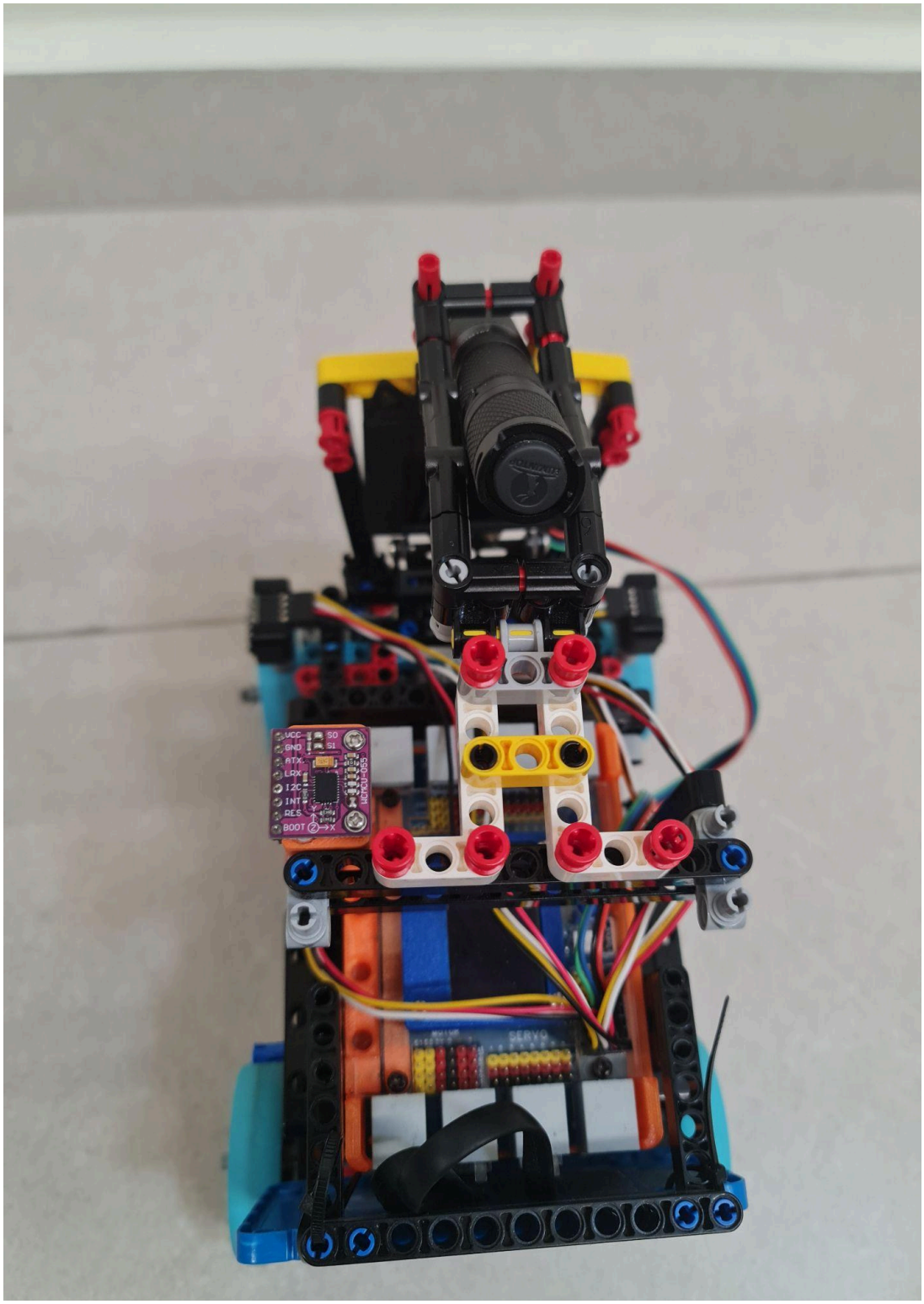


Front View:

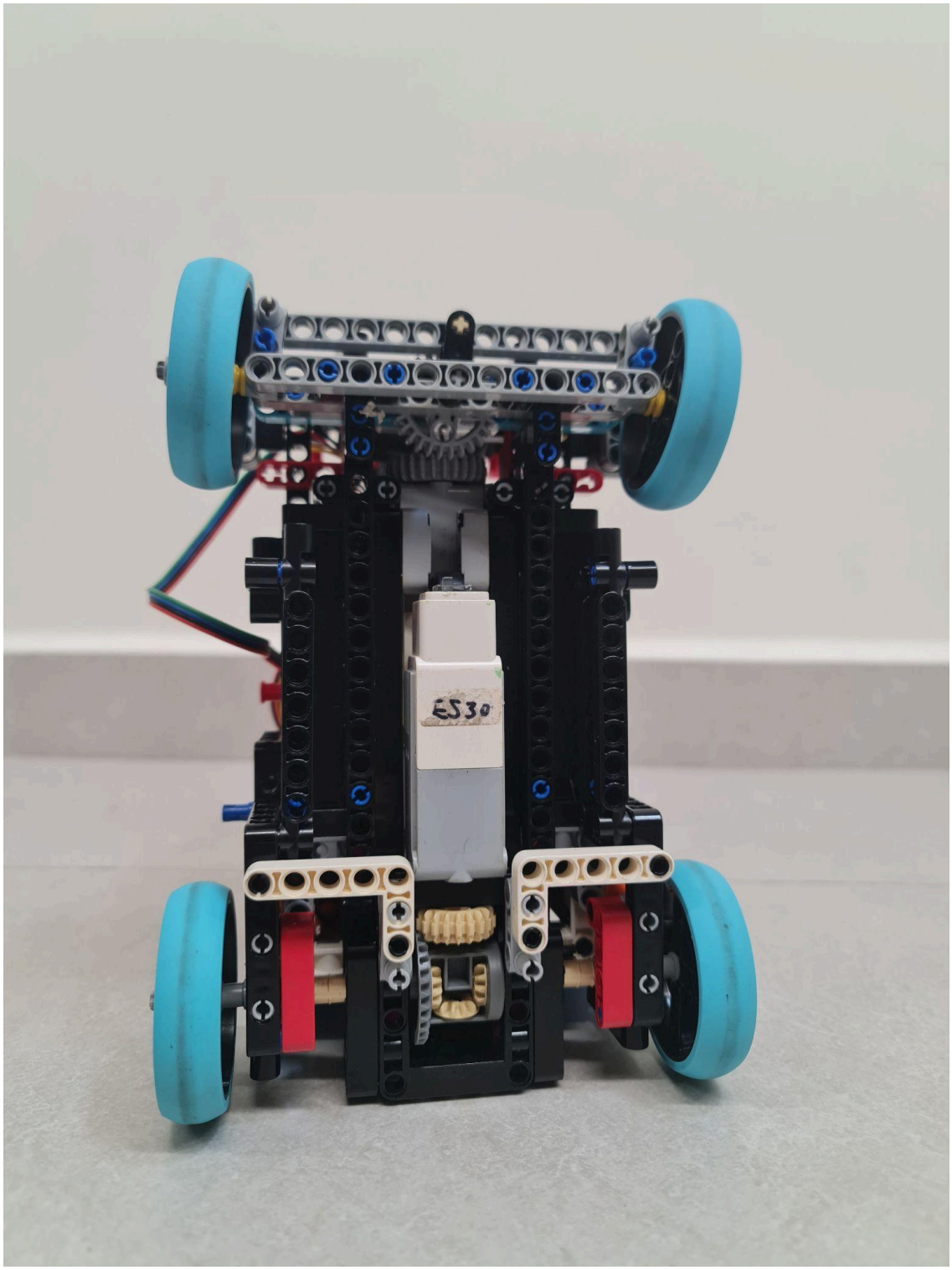


Back View:

Top View:

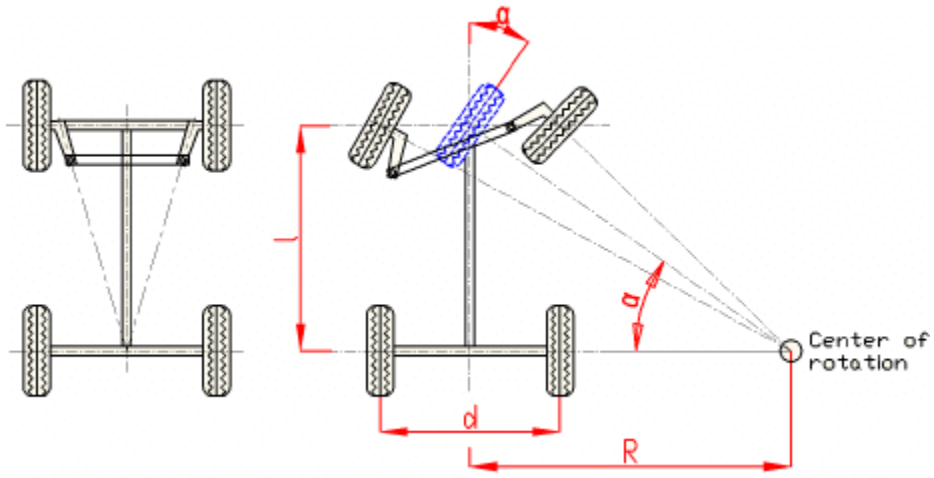


Bottom View:



Steering

Ackermann Steering



We chose Ackermann steering for our bot due to its ability to provide smooth, realistic, and mechanically efficient turning, especially for wheeled robots operating in constrained environments. Unlike differential steering, which relies on skidding or dragging wheels to turn, Ackermann steering mimics the geometry used in real-world vehicles by aligning the wheels to follow curved paths with minimal slip.

This steering geometry ensures that each wheel traces an arc of a common turning circle, reducing tyre wear and improving energy efficiency. It also allows for more precise control at both low and high speeds, which is critical for applications where stability and predictability of motion are important.

Moreover, implementing Ackermann steering offered us a valuable learning opportunity in replicating real automotive dynamics, preparing us for more complex robotic systems. The setup works especially well with our lightweight Lego chassis, as it maintains structural integrity while allowing realistic front-wheel steering.

Differential Gearing

The drive motor of our bot is connected to a differential gear system mounted at the base of the chassis. This system is crucial because it enables the left and right wheels to rotate at different speeds, which is essential for smooth and efficient turning. When a vehicle turns, the outer wheel must travel a greater distance than the inner wheel—without a differential, this mismatch would cause wheel slippage, increased friction, and strain on the motor. By allowing independent wheel speeds while maintaining a single motor input, the differential gearing ensures that our bot can navigate turns more smoothly and maintain better traction, especially on surfaces where precision and control are important. This setup improves both the handling and mechanical efficiency of the drive system, contributing to overall performance.

Motors

For our bot, we selected the EV3 Medium Motor as our primary actuator due to its optimal balance between speed, torque, and size. It provided sufficient power to drive our chassis while maintaining responsiveness and precision during manoeuvres. The compact form factor of the medium motor made it significantly easier to integrate into our Lego-based chassis without compromising the structural layout or adding unnecessary weight.

Initially, we explored the use of the EV3 Large Motor, which offers greater torque. However, during prototyping, we encountered several challenges with its larger physical dimensions. The bulkier design limited our flexibility in component placement, especially within the constraints of our compact bot structure. Additionally, while the large motor is capable of handling heavier loads, its slower rotation speed and weight made it less suitable for our application, which prioritised agility and moderate force output rather than brute strength. By using the medium motor, we achieved a desirable trade-off between performance and design efficiency, allowing our bot to move quickly, respond smoothly, and remain lightweight and modular.

Steering Motor

The steering motor is attached to the back of the bot. During testing, we discovered that placing the steering motor at the front caused the movement to become unstable and difficult to control, especially during turns. In contrast, positioning the steering motor at the back resulted in smoother and more consistent steering performance. This setup provided better alignment and improved the overall handling of the bot during navigation.

Drive Motor

The driving motor is responsible for propelling the bot forward and backwards. We positioned it centrally to ensure balanced power delivery to the wheels via the differential gear system. This motor plays a crucial role in determining the bot's speed and responsiveness. We selected a motor that provided sufficient torque for movement while still being lightweight and compact enough to fit within our chassis design. During testing, it delivered smooth acceleration and reliable performance, making it suitable for both straight-line movement and navigating turns in coordination with the steering mechanism.

Components

The Evolution X1 Controller Platform is a powerful and flexible development board designed to work seamlessly with the LEGO EV3 Mindstorms system. Powered by the ESP32-S3 microcontroller, this platform delivers a strong balance of connectivity, performance, and versatility. Below is an overview of its main features and the reasons we selected it for our project:

Key Features:

EV3 Mindstorms Compatibility:

The controller includes specialised connectors that support direct integration with EV3 motors and sensors. This feature allows us to reuse our existing EV3 hardware while benefiting from the advanced capabilities of the ESP32-S3.

Expanded GPIO Pinouts:

Beyond the EV3 connectors, the board provides a wide range of GPIO pinouts. These additional connections enable us to interface with a variety of external sensors and actuators, giving us greater flexibility for experimentation and customisation.

High-Performance ESP32-S3 Microcontroller:

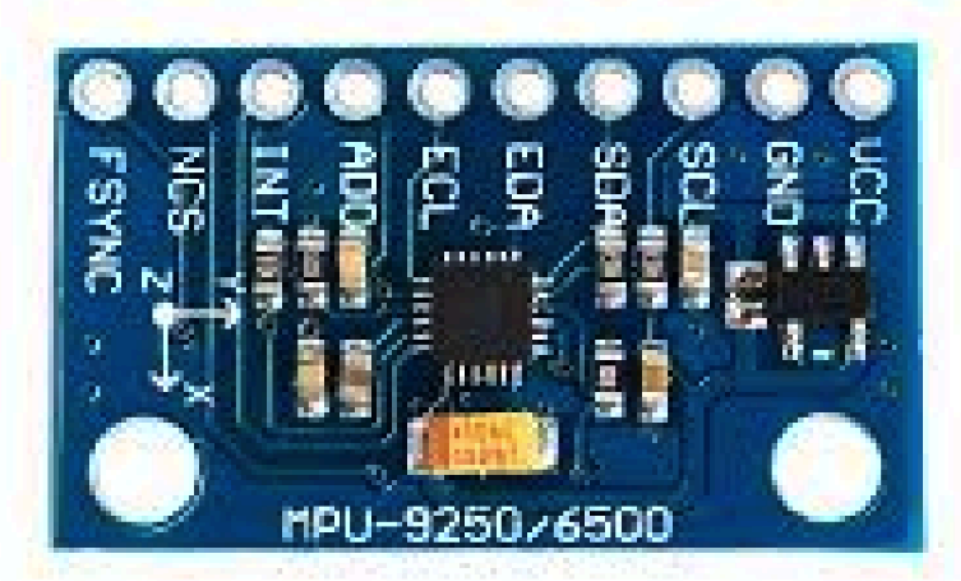
At the heart of the platform is the ESP32-S3, a highly capable microcontroller known for its efficiency and rich feature set. It includes dual-core processing, built-in Wi-Fi and Bluetooth, and supports many peripherals, making it suitable for a wide range of applications.

Developer-Friendly Environment:

The platform supports popular development ecosystems such as Arduino, MicroPython, and Espressif's IDF, making coding more accessible. Its thorough documentation and active community support help streamline the development process and reduce technical barriers.

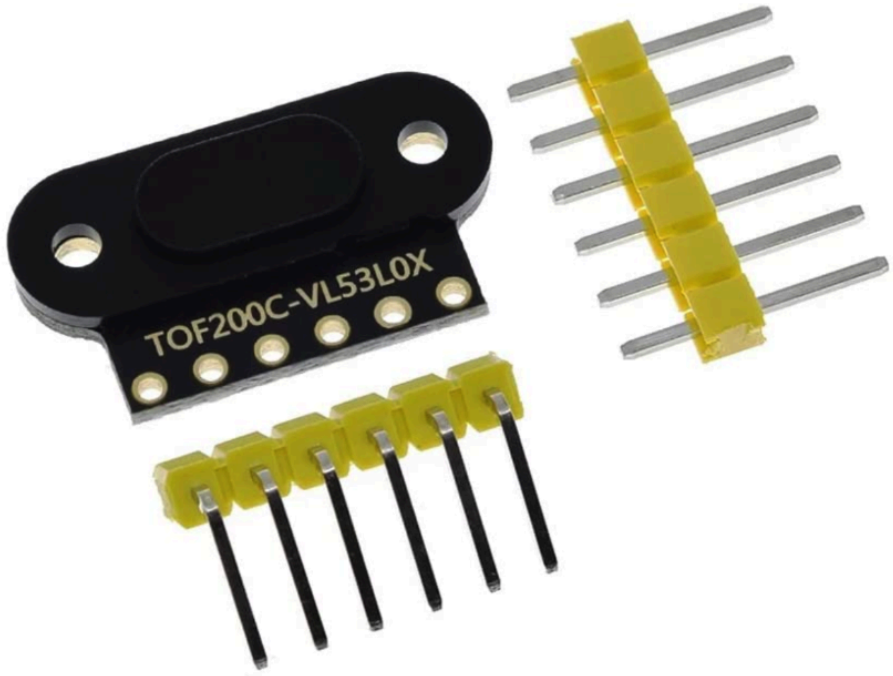
Gyroscope

For our gyroscope, we chose the EvoBNO055, an all-in-one 9-axis motion sensor that integrates a 3-axis gyroscope, 3-axis accelerometer, and 3-axis magnetometer, along with a built-in sensor fusion processor. We selected this module because it performs onboard sensor fusion using an internal ARM Cortex-M0, allowing it to output absolute orientation data such as Euler angles and quaternions, directly saving us the effort of implementing complex fusion algorithms ourselves. This greatly reduces software complexity and improves real-time performance. The gyroscope and accelerometer provide reliable measurements of angular velocity and linear acceleration, while the magnetometer adds heading information, making the sensor ideal for accurate motion tracking and navigation. Additionally, the EvoBNO055 supports standard communication protocols like I2C and UART, which integrate well with our control system.



Time of Flight (ToF)

For our distance sensing, we selected the EvoVL53L0X, a compact Time-of-Flight (ToF) sensor based on ST's VL53L0X module. This sensor works by emitting infrared light and measuring the time it takes for the light to bounce back from an object, then converting that into distance. We chose it because it offers accurate ranging up to approximately 2 meters, a significant improvement over traditional IR sensors—while being largely unaffected by surface colour or texture. It delivers a fine 1 mm resolution, and typical measurements fall within ±3% accuracy depending on conditions. Additionally, it operates across a wide voltage range (approximately 2.6–5.5 V) and includes on-board level shifting and a voltage regulator, making it simple to integrate with our EvolutionX1 controller. Communication is handled via I^2C, with options for hardware shutdown (XSHUT) and interrupt signalling (GPIO1), providing both flexibility and safety in system design

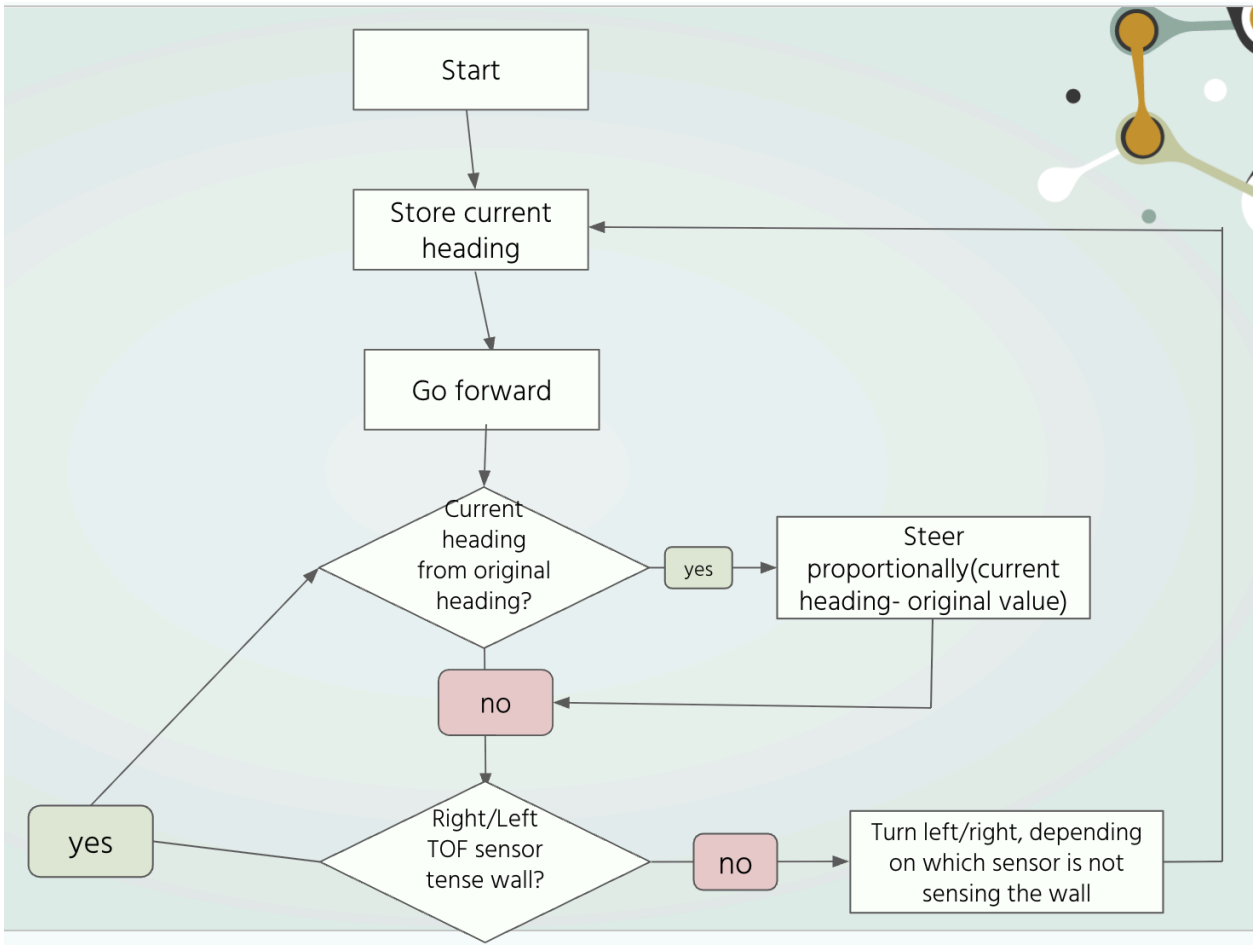


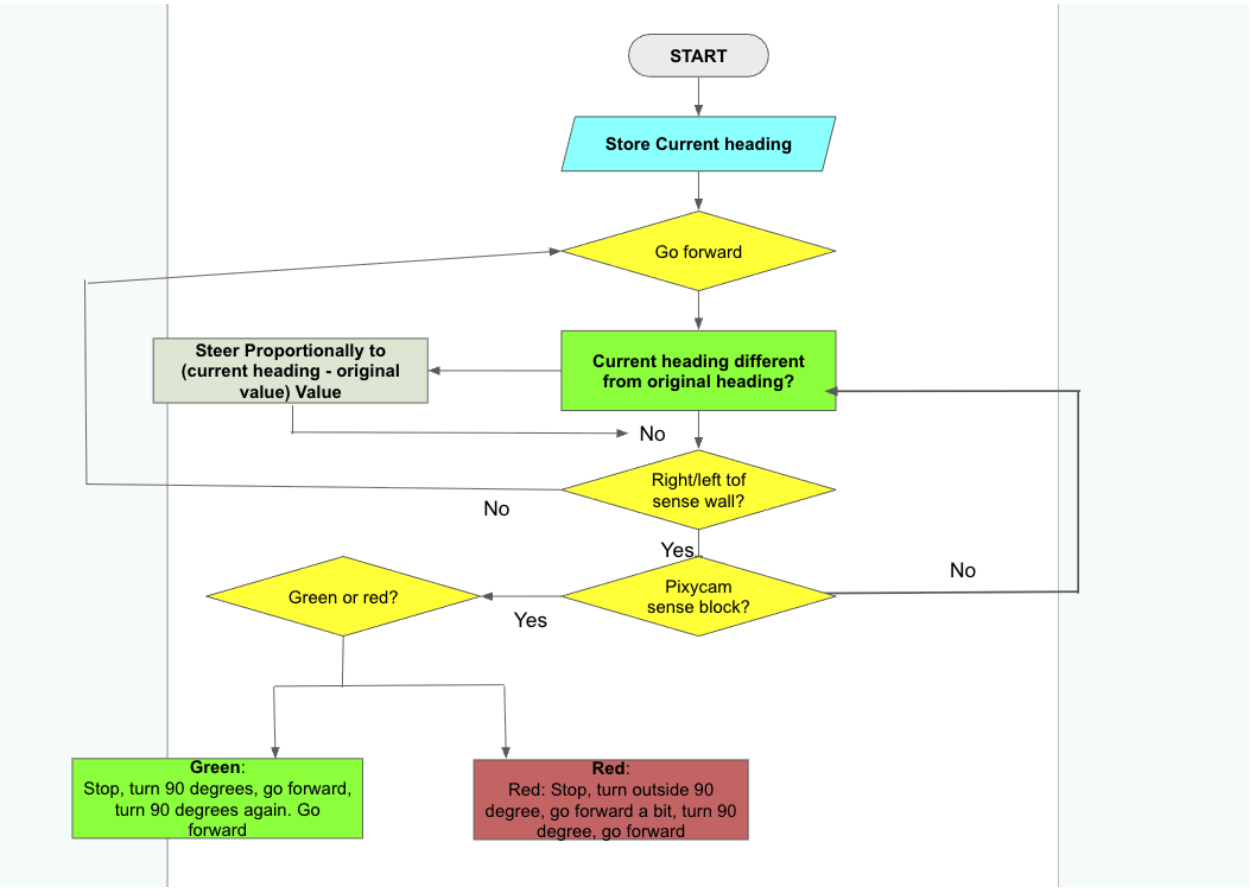
Huskylens

The HuskyLens is an AI-powered vision sensor capable of recognising objects, colours, faces, and more. It comes preloaded with various AI algorithms such as object tracking, face recognition, and colour recognition, which can be used directly without the need for complex coding. In our project, we used the HuskyLens specifically to detect coloured blocks, which is critical during the obstacle and navigation challenges. One of the key reasons we selected the HuskyLens is its built-in ability to learn and identify different colours accurately, making it ideal for our task. Additionally, the sensor includes a built-in screen on the back, which displays real-time visual feedback, greatly simplifying the training and testing process. Lastly, its compact and lightweight design makes it easy to mount onto our bot, allowing for seamless integration without adding bulk.

Strategy

Flowchart for Preliminary Round





Code

To ensure that every team member could easily understand the code, we made readability our top priority. We ensured that all parts of the code were clearly labelled with comments to explain their purpose. This approach made it much easier to break down the complexity of our autonomous robot system into manageable sections.

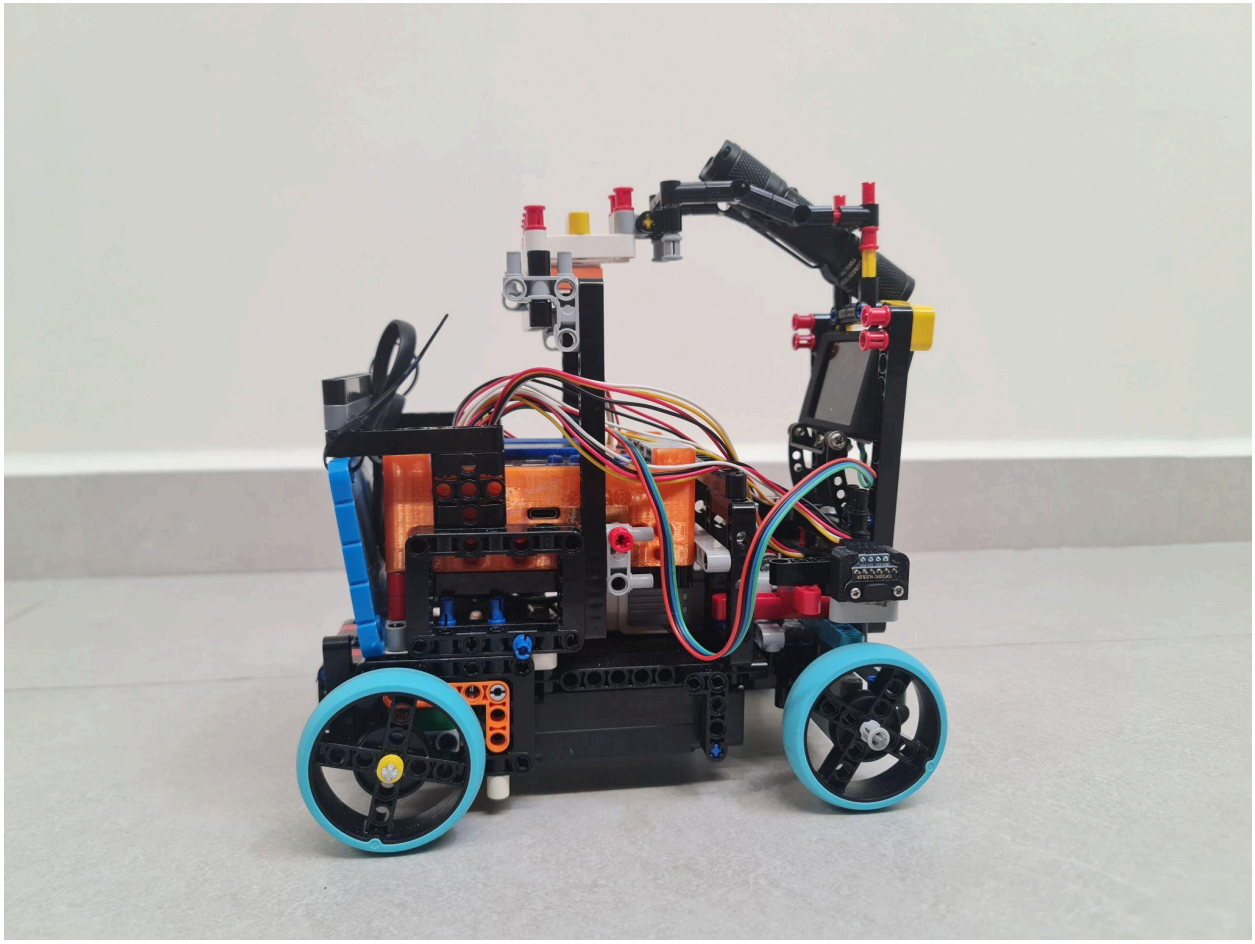
1. To maintain code clarity and organisation, we followed these practices:

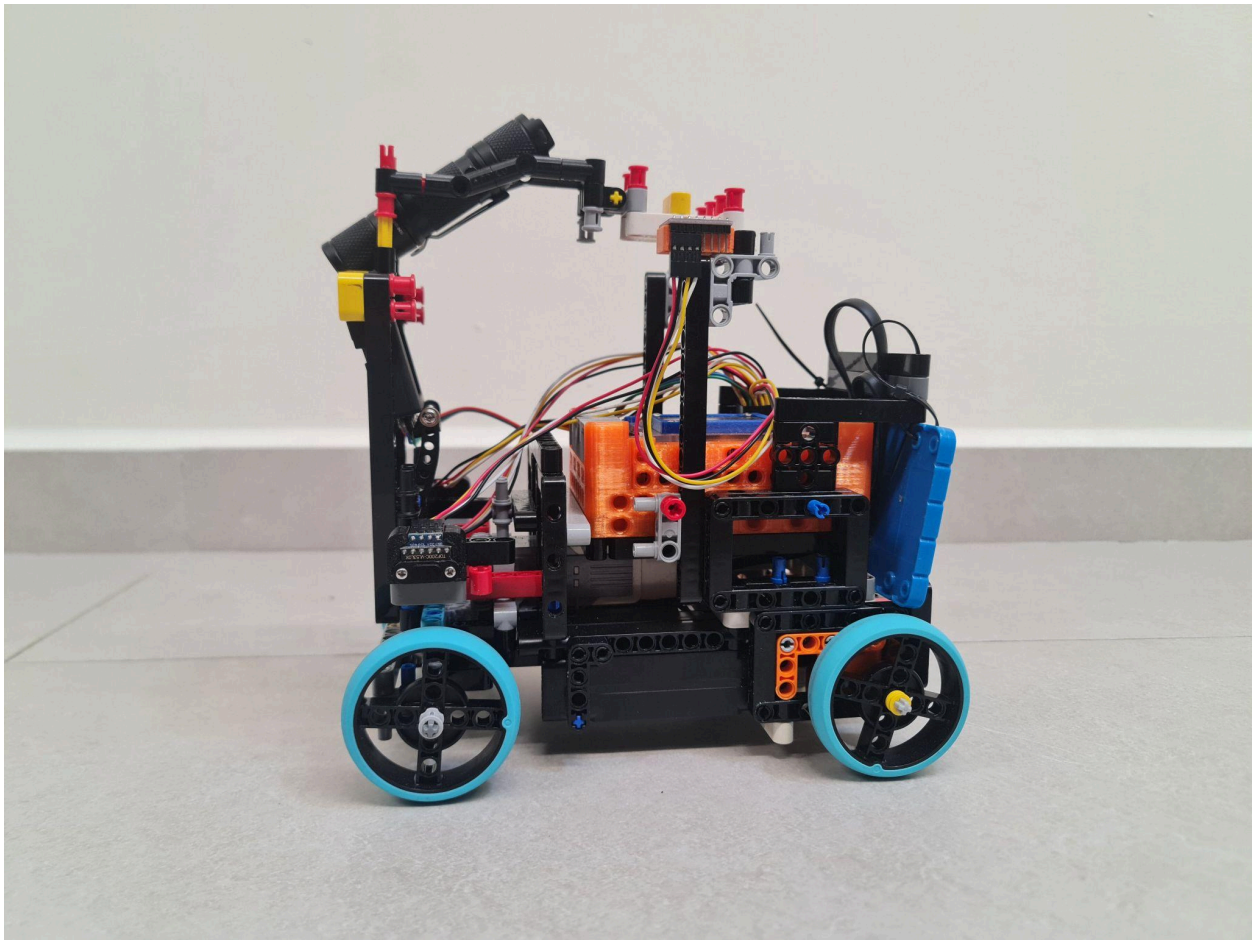
2. Added descriptive comments throughout the code to explain each section
3. Declared large code segments (such as functions) at the start for easy reference, but placed their full implementations at the bottom
4. Made use of libraries and custom classes to keep the main script clean and modular
5. Used clear and meaningful names for variables and functions to reflect their roles and purposes

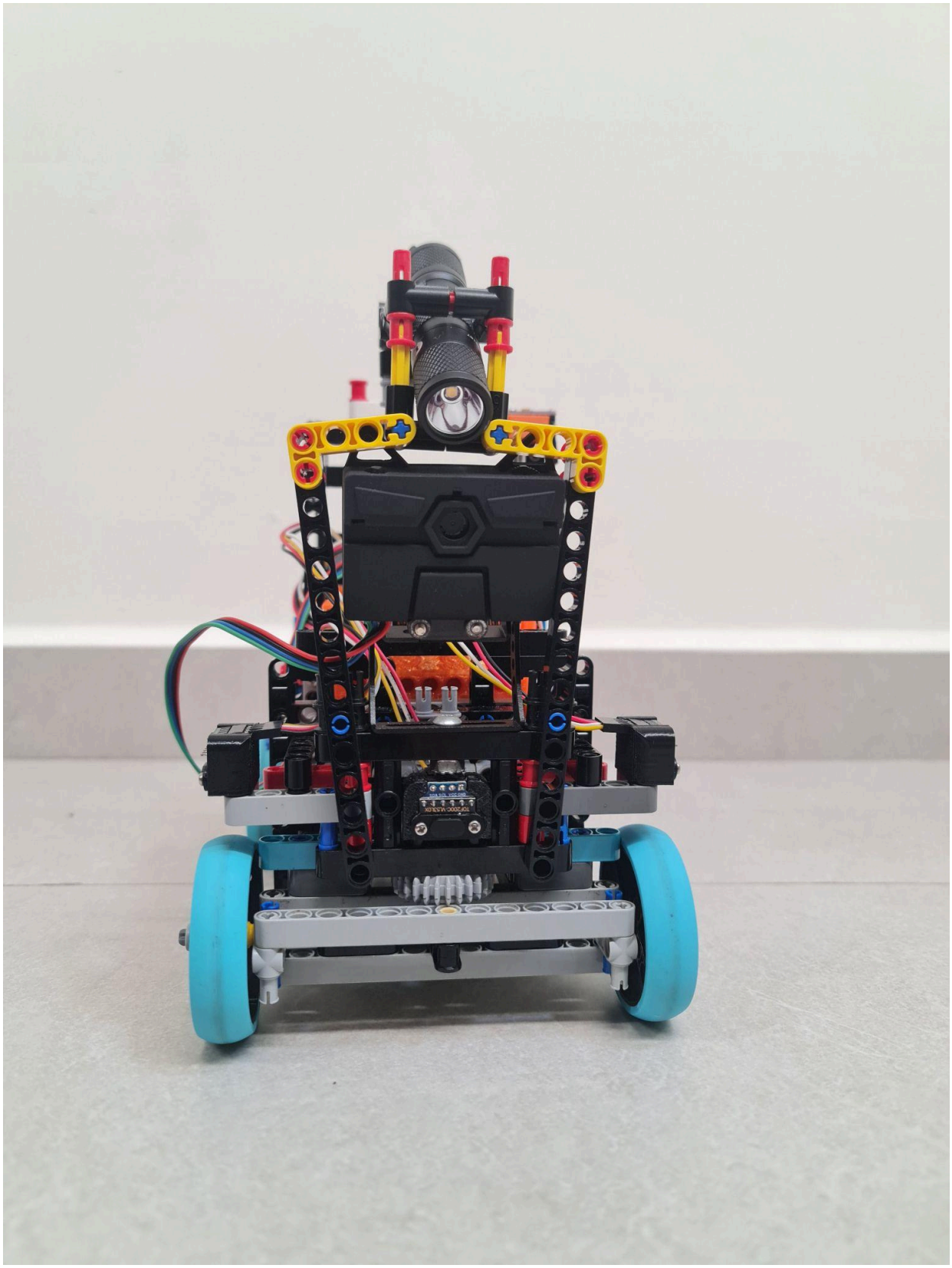
Photos

Formal Photo

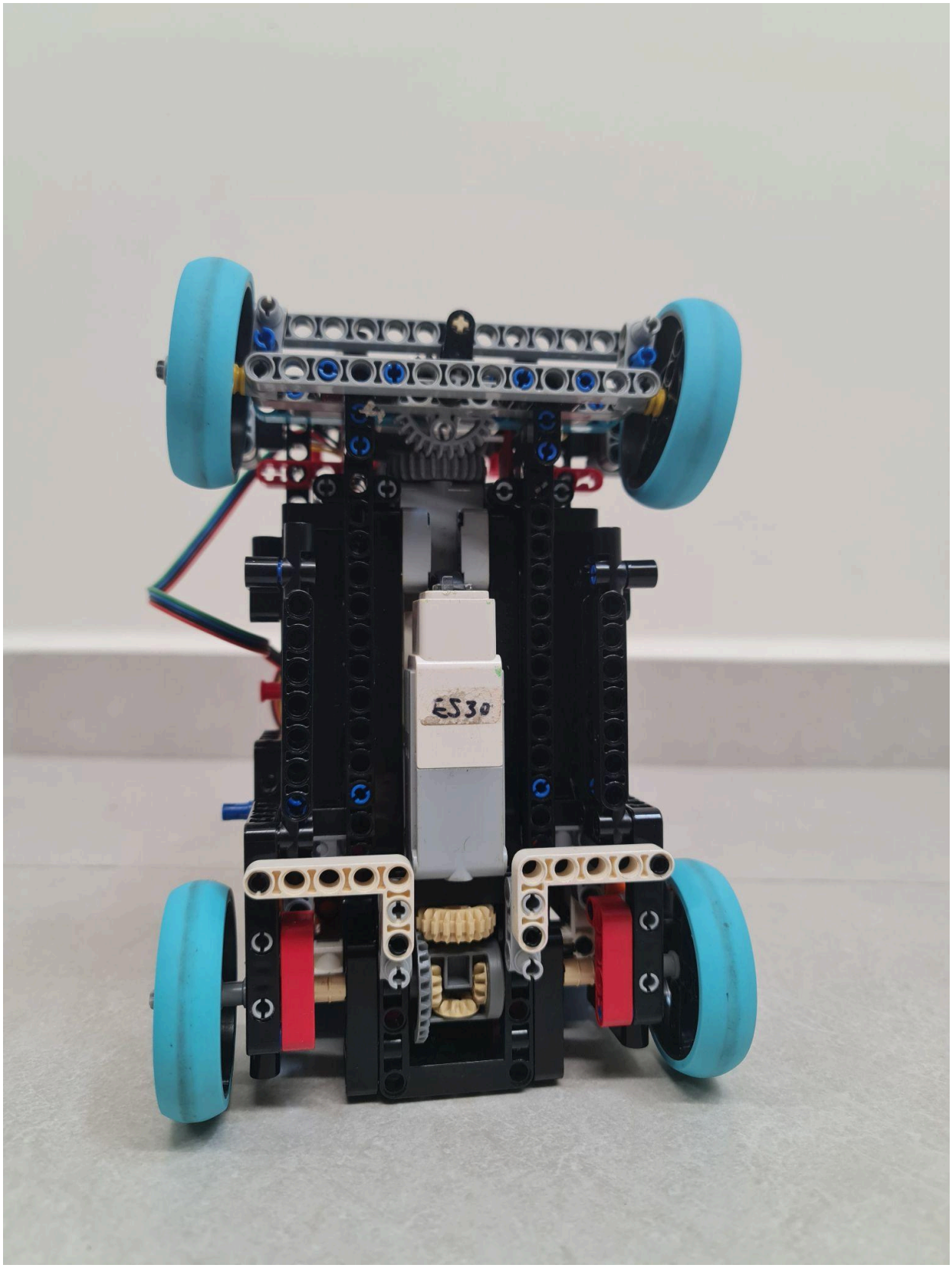
Informal Photo

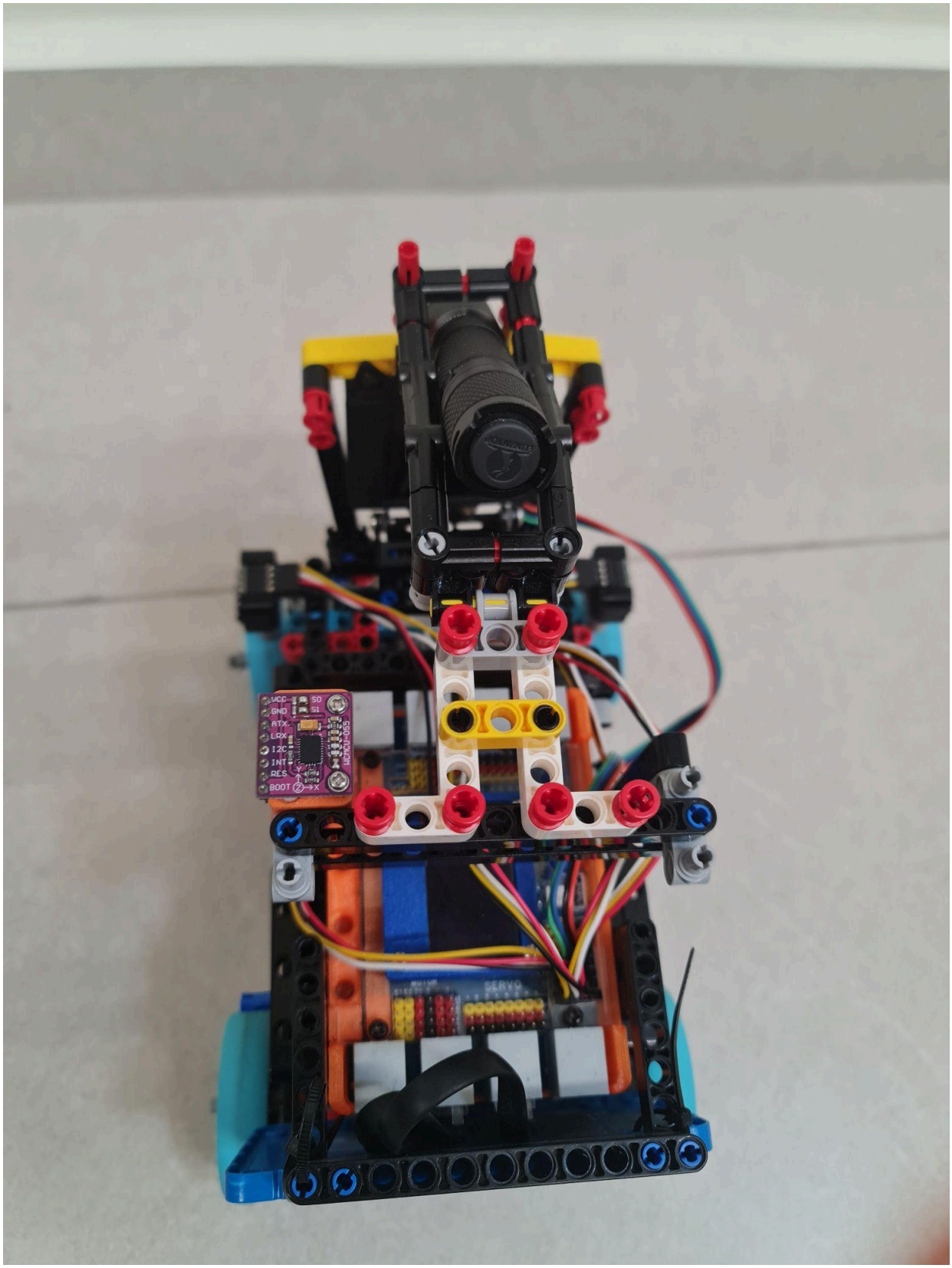







HOG
RIDA
VEX


ES30


VCC
GND
RTX
LRX
I2C
INT
RES
BOOT
SO
SI
SERVO