EE-10 Tembo: Autonomous Delivery Robot

University of Nottingham
UK | CHINA | MALAYSIA

EE-GROUP 10
Muhammad Nazrin bin Aminuddin
Nathan Mathias Lukambinga

Background

Project Objective:
Construction of **Raspberry Pi**-powered autonomous delivery robot prototype programmed with **Python 3** and **OpenCV**.

Prototype Requirements:
- Robot follows the designated track with acceptable error.
- Robot recognizes shapes, arrows and traffic signs while moving on track.

Delivery Milestones:

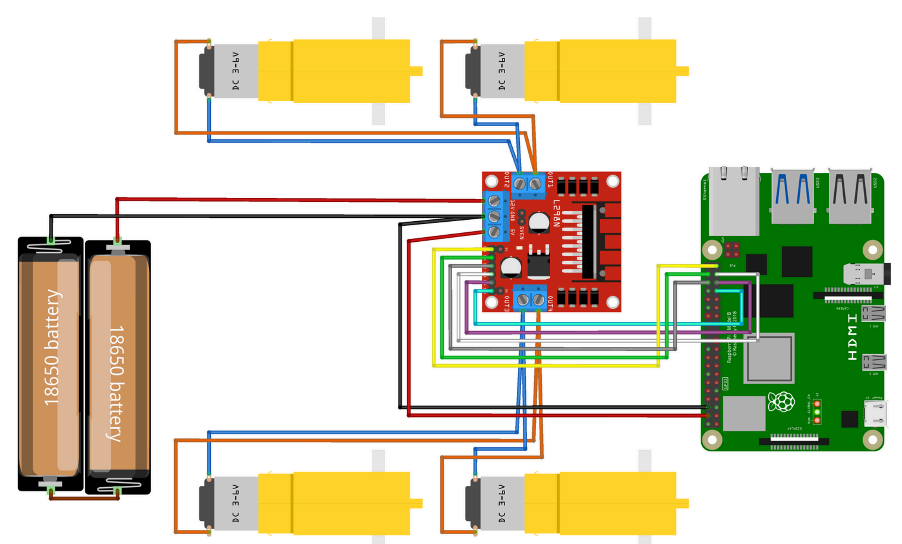
Project Week 1:
- Raspberry Pi Setup
- Movement Control System

Project Week 2:
- Basic Line Following System (Black)
- Shape and Arrow Detection System
- *Additional* : Facial Recognition System

Project Week 3:
- Advanced Line Following System (Multiple Colours)
- Systems Integration
- Prototype Showcase (Track Challenge)

Movement Control System

Drivetrain Architecture
Consists of 4 geared DC motors controlled by an L298N dual H-Bridge motor driver in a **four-wheel drive (4WD)** system.



Locomotion Control Experiments (Project Week 1)

Angle Turning:
- Robot can perform either left or right turns at 30, 45, 60 and 90 degree angles.
- Specific angles can be configured by adjusting a time delay function for the turning motions.

Distance Measurement:
- Robot can measure total distance it has travelled using rotary encoders attached to motors on both sides.
- Distance can be obtained by the following equation:

$$Distance = \frac{Total\ number\ of\ pulses}{Pulses\ per\ revolution} \times Circumference$$

Remote Motion Control:
- Robot can be wirelessly controlled using VNC to perform basic maneuvers, change speeds and execute functions.

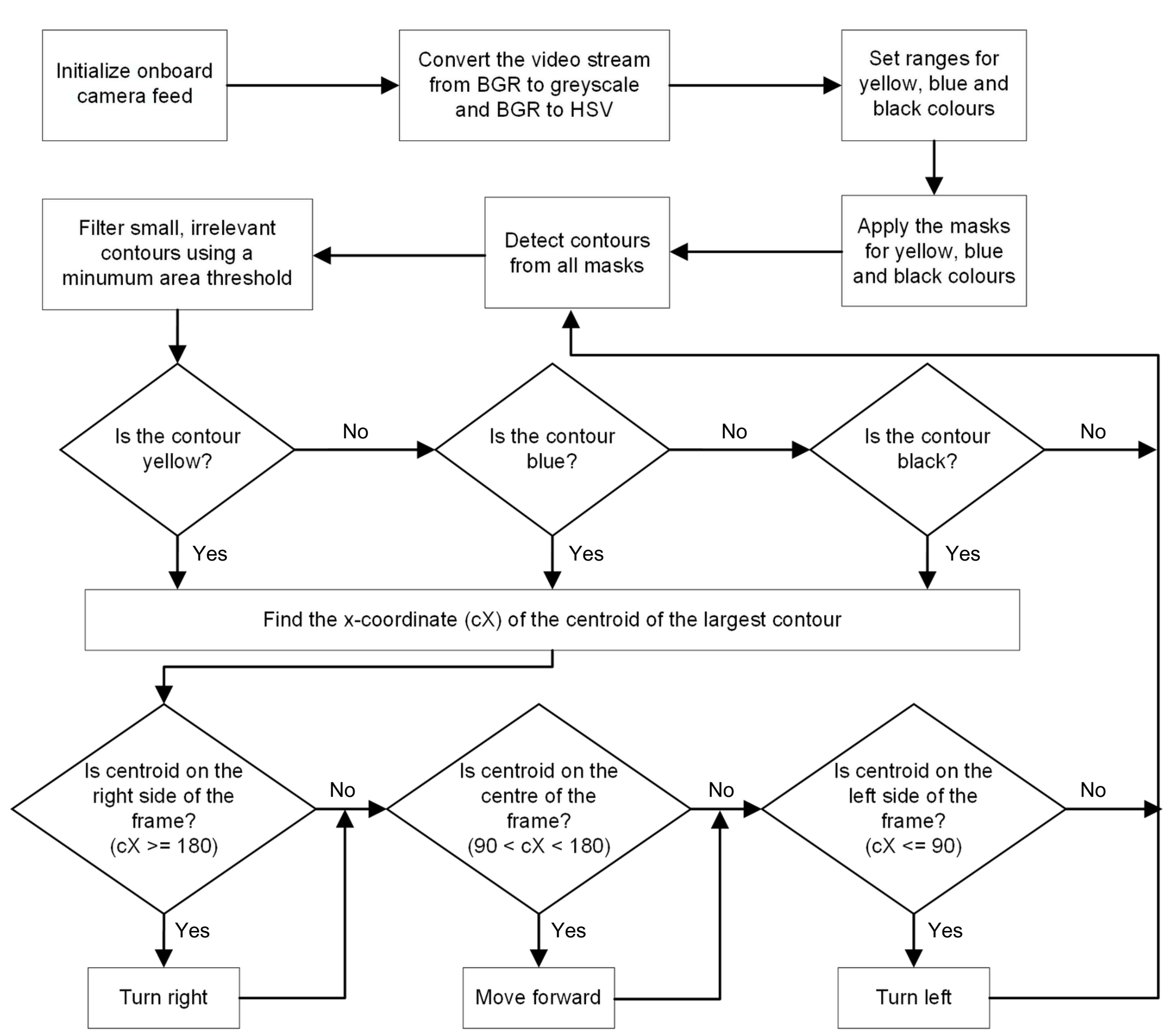
Line Following System

System Approach

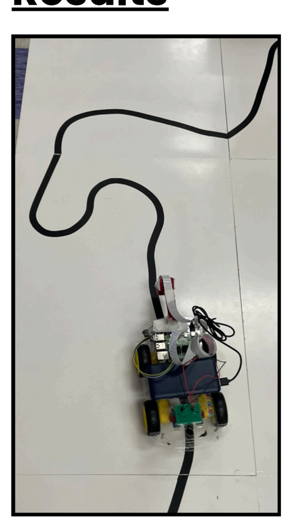
Image Processing:
- Original video stream (RGB) converted into **Greyscale** and **Hue-Saturation-Value (HSV)** formats to facilitate image colour consistency and ease segmentation.
- Create **masks** for all assigned track line colours to isolate areas that fall within specified ranges:
  - **Black:** [(0, 0, 0), (70, 70, 60)]
  - **Yellow:** [(20, 100, 100), (40, 255, 255)]
  - **Blue:** [(100, 75, 2),(100, 255, 255)]

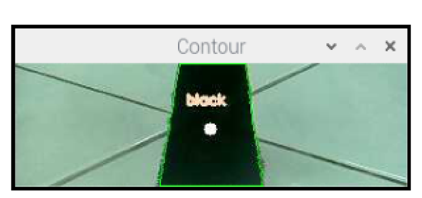
Contour Analysis:
- **"cv2.findContours( )"** function detects the line contours in each colour mask.
- The moments of the contour are calculated to locate the **centroid** (reference point).
- The direction of the track is determined based on the x-coordinate position of the centroid.

Methodology

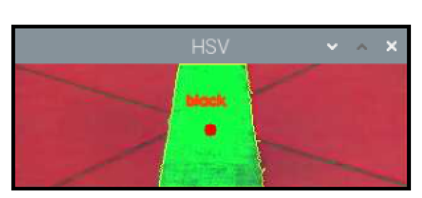


Results

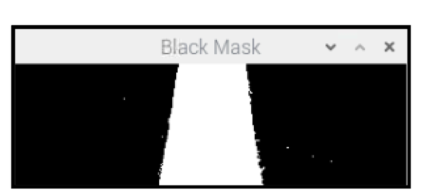




Original Contour Frame



HSV Contour Frame



Black Mask Frame

Blue Mask Frame

System Refinements
- Executing all movements within an optimum duty cycle range of 30% to 40% for best track performance.
- Resizing the video frame to a **Region of Interest (ROI)** frame (320x100 px) to focus on specific area of the track.

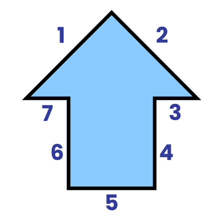
Shape Detection System

System Approach

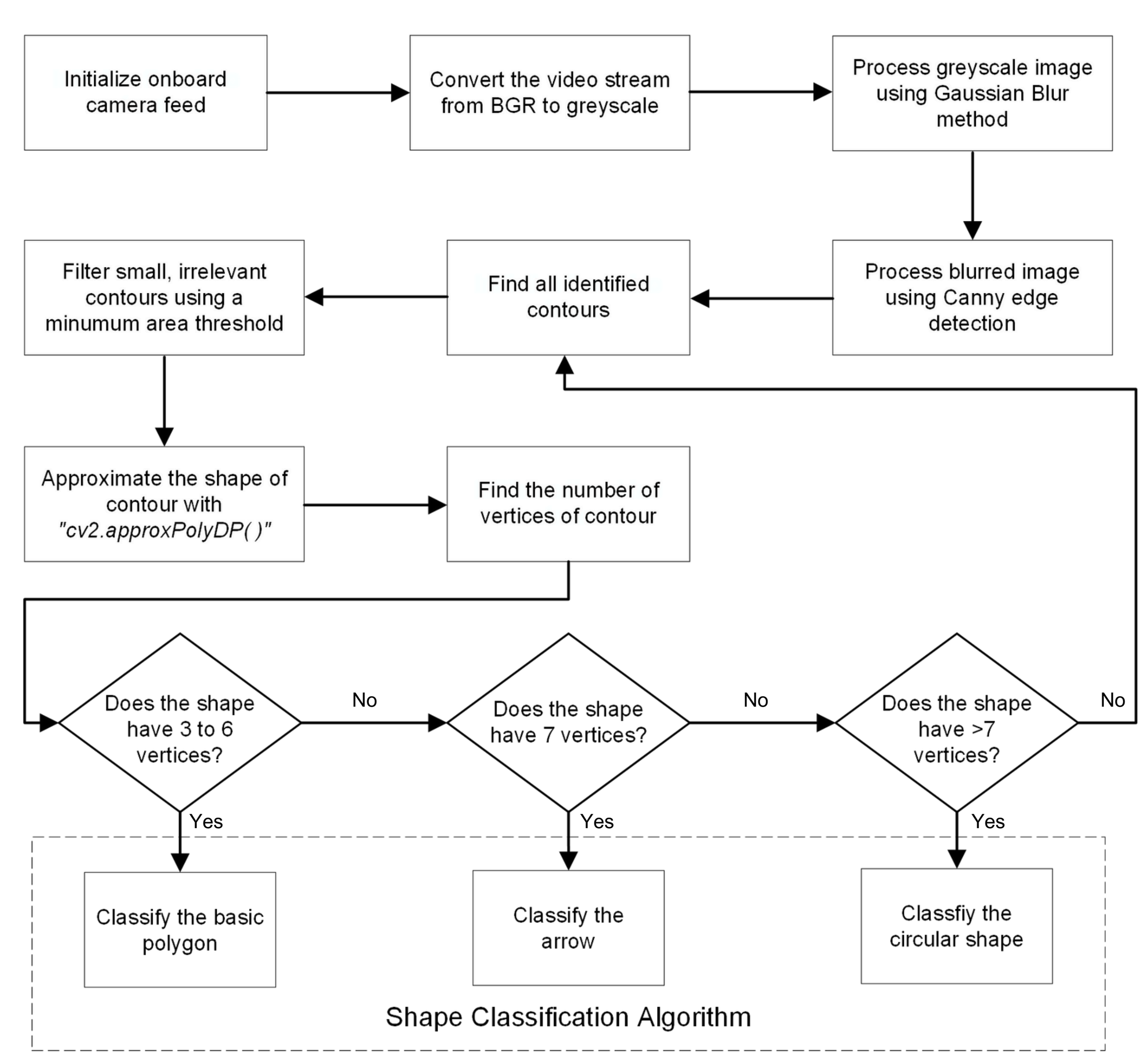
Image Processing:
- Original video stream (RGB) converted into **Greyscale** format to reduce the image colour complexity.
- Greyscale video stream undergoes a **Gaussian Blur** operation to reduce the image noise and details.

Contour Analysis:
- **Canny edge detection** algorithm is extracts edge features from the processed image.
- **"cv2.findContours( )"** function retrieves all identified contours from the Canny output.
- **"cv2.approxPolyDP( )"** function approximates the contour shape by simplifying the number of vertices.

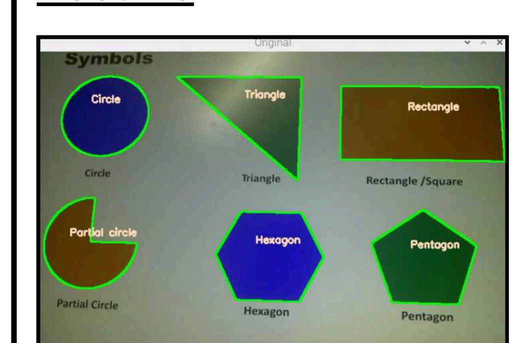
Shape Classification Algorithm:
- **Basic Polygons**
  - Shape has between 3 to 6 vertices.
- **Arrows**
  - Shape must have 7 vertices.
  - Identify arrow direction by position of its base and tip.
- **Circular Shapes**
  - Shape has >7 vertices.
  - Compare the shape area to area of a circle ($A=\pi r^2$).

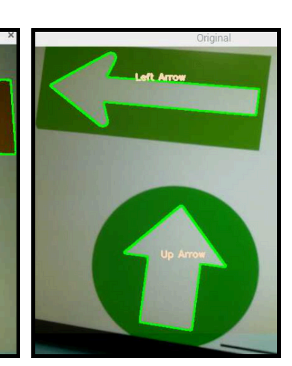


Methodology

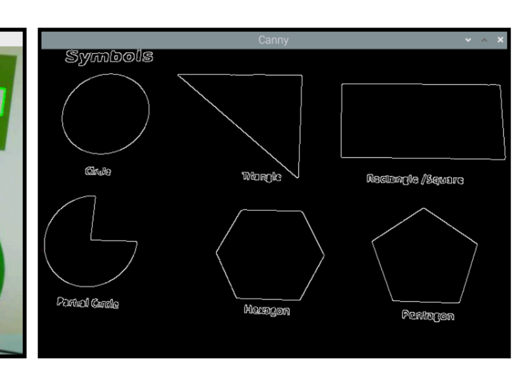


Results





Original video stream frames with detected shapes and arrows.



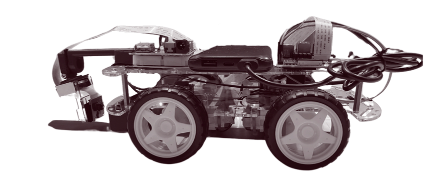
Canny edge detection frame with white edge features.

Systems Integration

Purpose & Approach
Robot has ability to execute multiple tasks simultaneously on-track.
- Merger of source codes for the **Line Following**, **Shape Detection** and **Movement Control** systems.
- Implementation of integrated **Image Processing** and **Video Display** functions for the OpenCV-based systems.

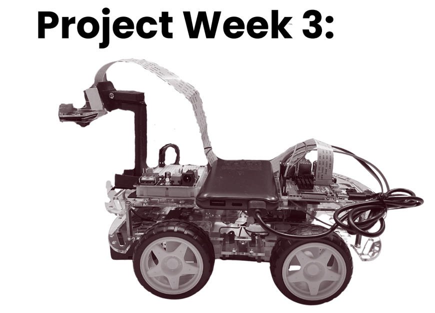
Chassis Design
Evolution of chassis configuration to adapt to new track challenges.

Project Week 2:



- Camera points inwards towards the undercarriage of the robot.
- Accurate movements in tight track corners and chicanes.
- Unable to detect on-track shapes properly due to camera angle.

Project Week 3:



- Camera positioned higher above the robot and points downwards.
- Video feed captures larger field of view of the track environment.

Practical Challenges
Robot does not recognize shapes and arrows while moving on-track with consistent reliability.

Identified Issue:
- Robot moves too fast for the video stream to capture a full image of any identifiable object.

Proposed Solution:
- Robot temporarily stops movement when it identifies any non-monochrome colours on-track (lines or objects) to allow a stable video stream for detection processing.

Conclusion

Accomplishments:
- Achieved most of the delivery milestone targets set throughout all three project weeks.
- Completed minimum requirements during on-track prototype showcase with acceptable error.

Future Improvements:
- **Track Sign Detection System** using the template matching approach within OpenCV. Allows the robot to recognize and respond to the available track signs.
- **Proportional-Integral-Derivative (PID) Controller** implementation to refine the Line Following System. Enables smoother and more precise movement while steering on the track.
- **Unified User Interface (GUI) Dashboard** to display real-time information (i.e. video feeds, on-track telemetry data) or configure the robot control settings (i.e. trackbars). Can use Python tools such as Tkinter or PySide.