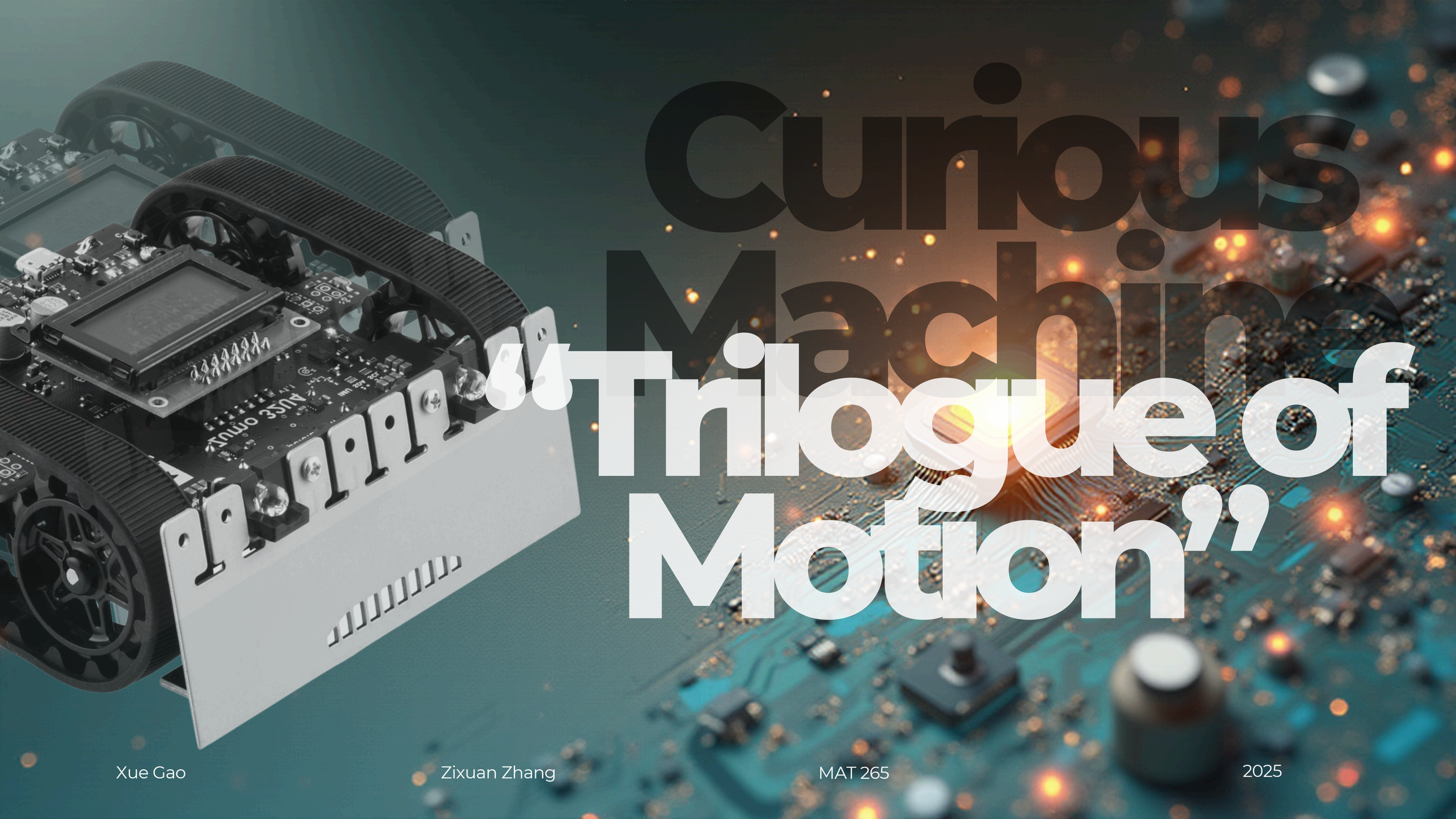


“Trilogue of Motion”

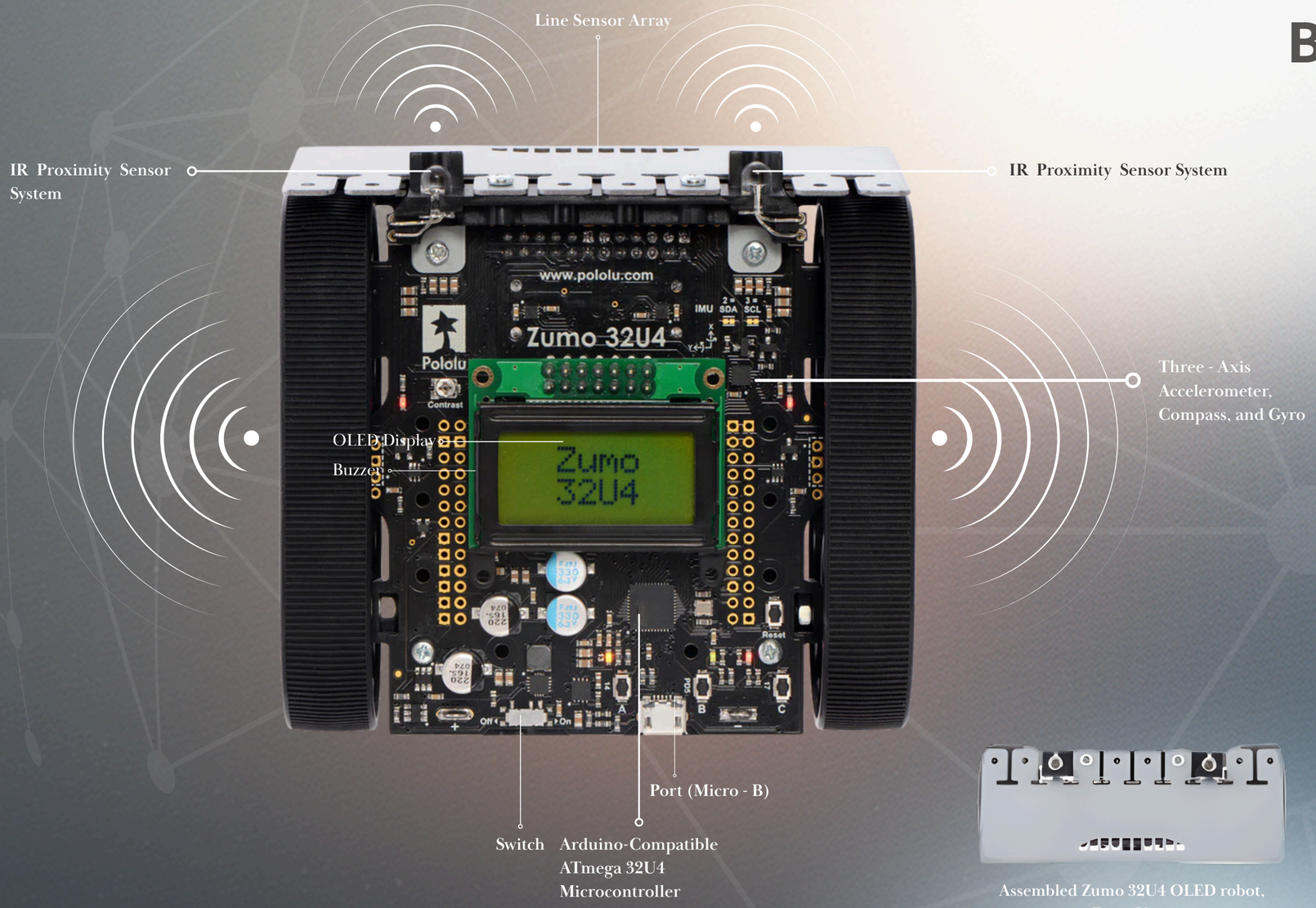
Xue Gao

Zixuan Zhang

MAT 265

2025

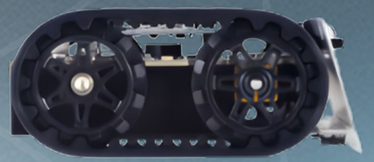
Understanding Zumo 32U4



Bullet Point

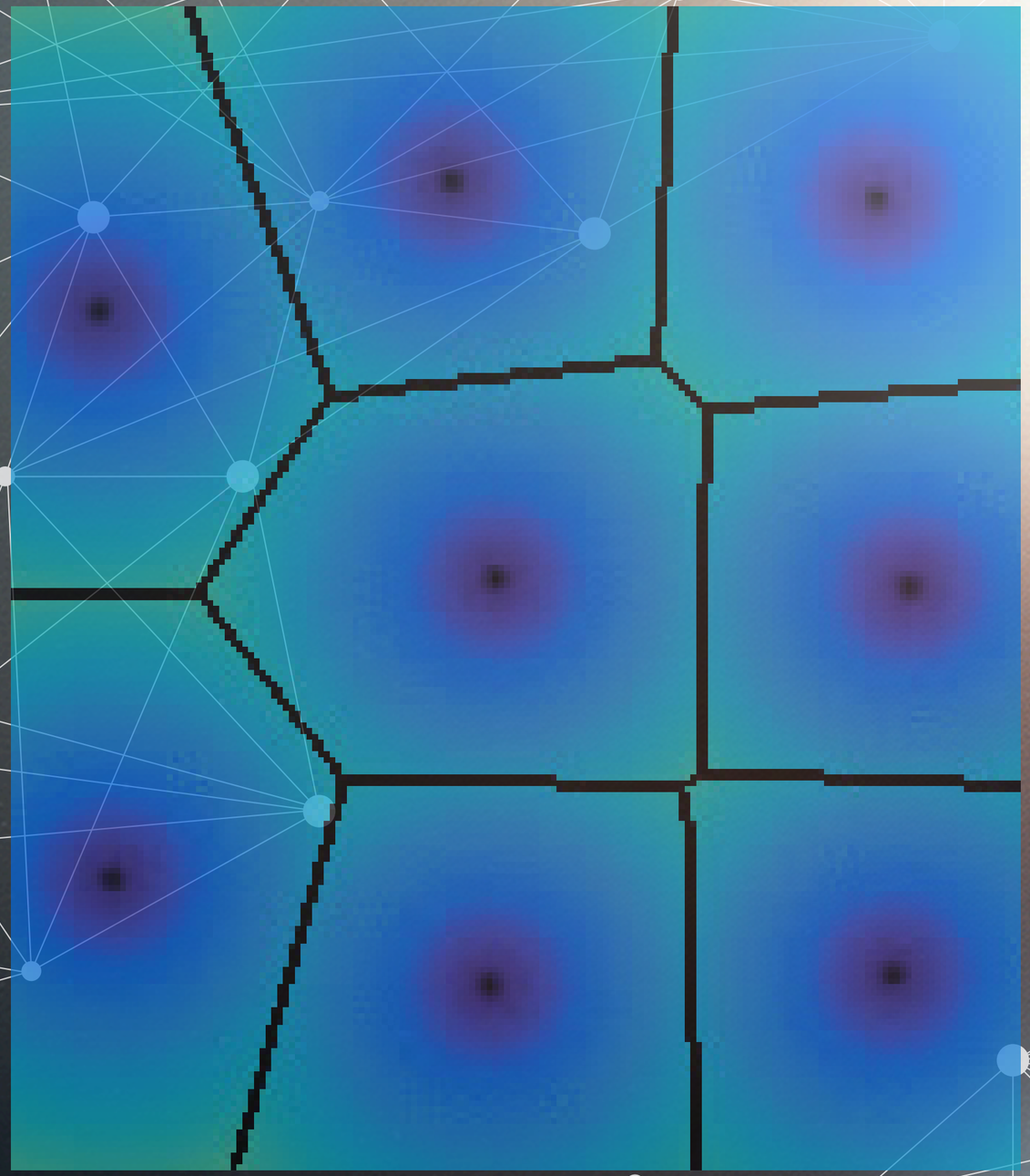
- Motors & Encoders: measure distance and direction.
- Line Sensors: detect light/dark surfaces for navigation.
- Proximity Sensors: sense nearby objects (30–40 cm range).
- Buzzer / LED / LCD: provide auditory and visual feedback.
- Programming via Arduino IDE: libraries installed through package_pololu_index.json

Assembled Zumo 32U4 OLED robot, Front View



Assembled Zumo 32U4 OLED robot, Side View

Assembled Zumo 32U4 OLED robot, Back View



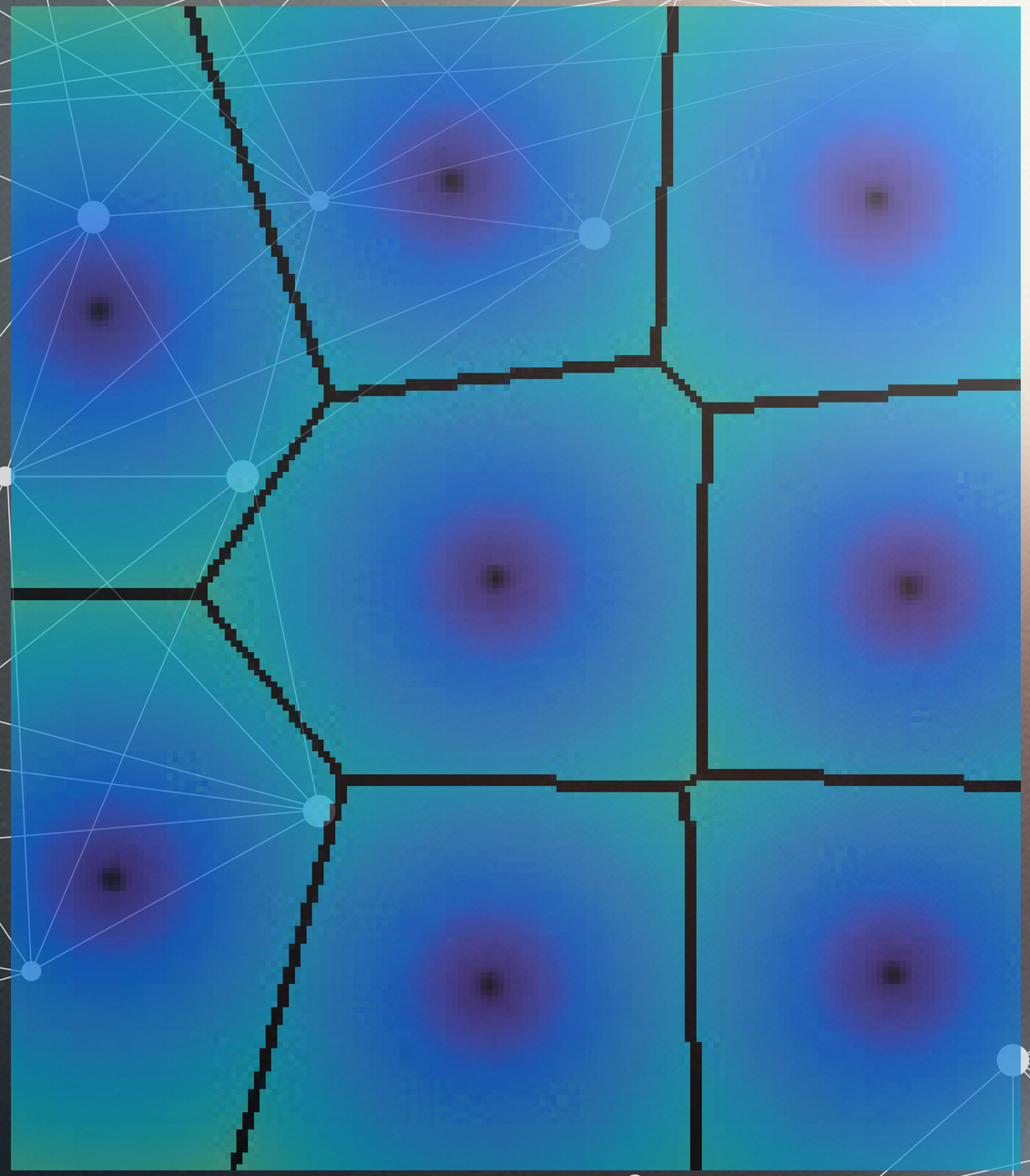
Concept Develop

Voronoi Diagram

- Divides space into regions around each point.
- Each region represents all locations closest to that point.
- Visualizes how individuals claim and negotiate space.

Delaunay Triangulation

- Connects points whose Voronoi regions touch.
- Creates triangles that form the most stable network between neighbors.
- Represents communication and structural balance among entities.



Trilogue of Motion

Project 1(Outcome of MAT265)

Machine vs. Machine

The three autonomous machines navigate a shared space, forming a living geometry in motion.
Guided by the principles of Voronoi and Delaunay, they continuously adjust their positions—expanding, contracting, and reorienting as one interconnected structure.
Their movement follows a logic of distance symmetry and spatial equilibrium, creating a collective form of motionwhere geometry and interaction define the behavior of this emerging mechanical species.

If we think the robots as a species...

“Who is my own kind?”

“How do I interact with my own kind?”
distance, angle...

We start with an exploration of 2-robot relationship...

What We Achieve So Far?

Line Sensors:

- **Face downward**
- **Light and dark surfaces.**
- **Sources of infrared light, like the sun.**

Proximity Sensors:

- **Face in different directions away from the Zumo**
- **Detect nearby objects.**
- **Commands from typical IR remote controls.**
- **ONLY Detect reflected IR light that is turned on and off quickly at a frequency of 38 kHz.**
- **Effective sensing range is about 30–40 cm**

Problem That We Have?

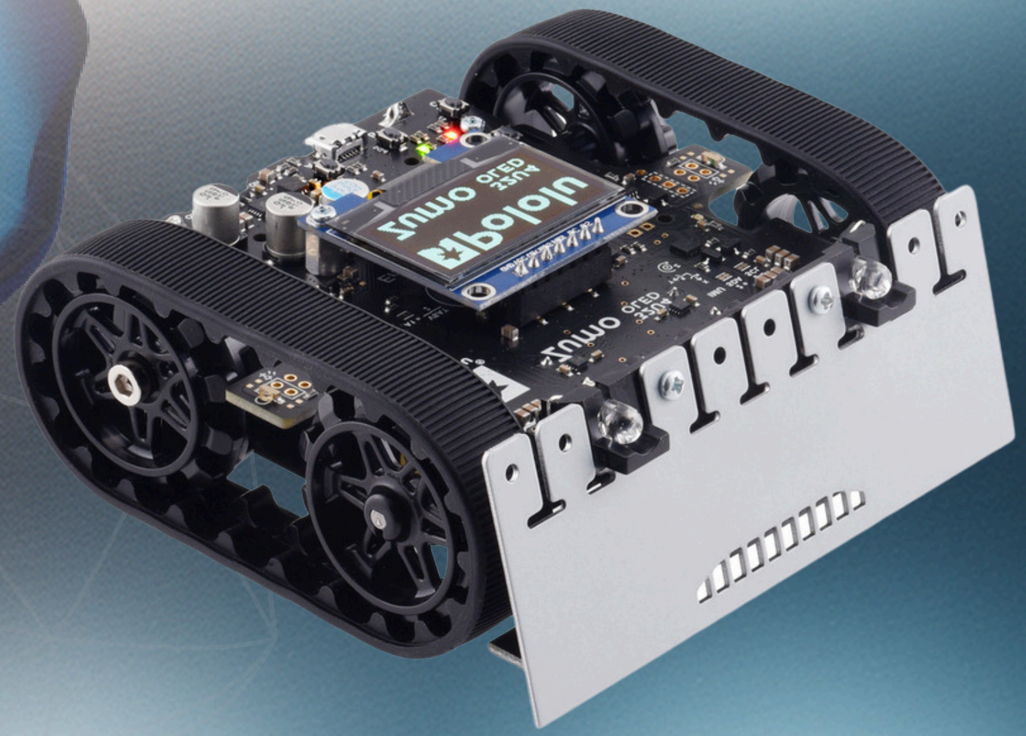
We found that the maximum sensing distance was around 30 cm to 40 cm, with a limitation of around a 50-degree angle.

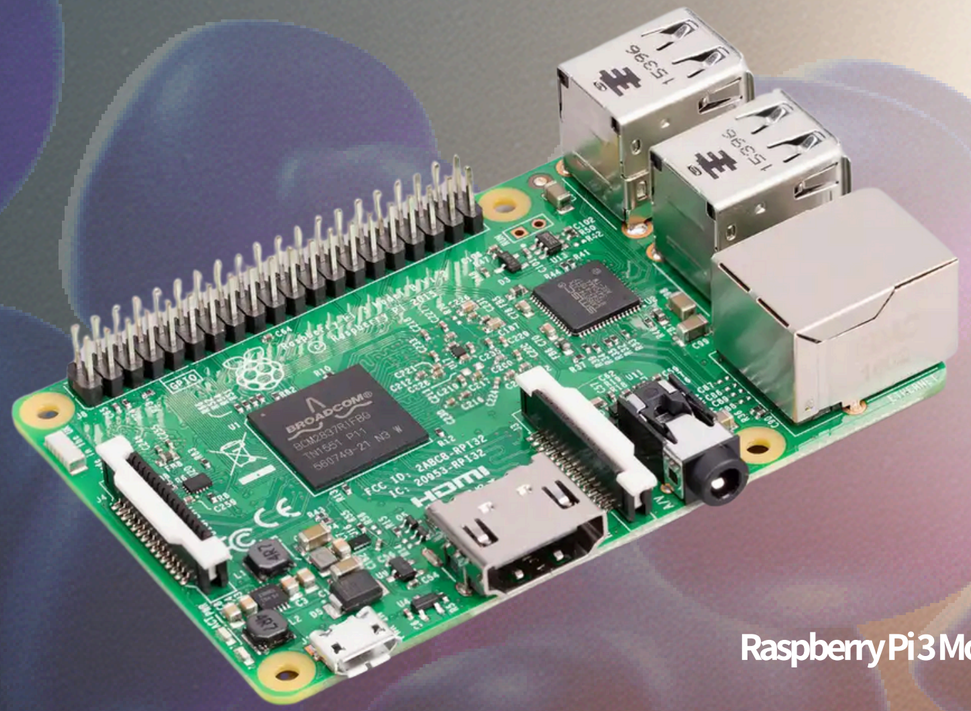
There is a significant dead spot between the sensing regions of the front sensor and each side sensor. Therefore, if the Zumo senses an object with the left or right sensors and then turns to face it, there will probably be a period of time where none of the sensors can see the object.

Solution 1

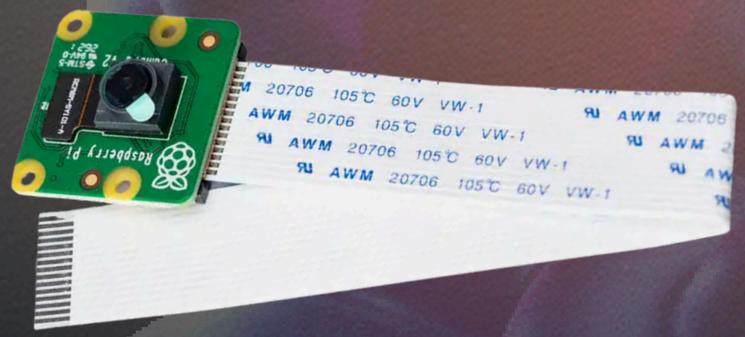
Sensors within Zumo system

By adding Sharp Distance Sensors, which use infrared triangulation instead of simple reflection, the robots can measure distance more accurately and continuously over a wider range (typically 10 cm – 80 cm). These sensors provide analog distance data rather than a simple “object detected” signal





Raspberry Pi 3 Modle B



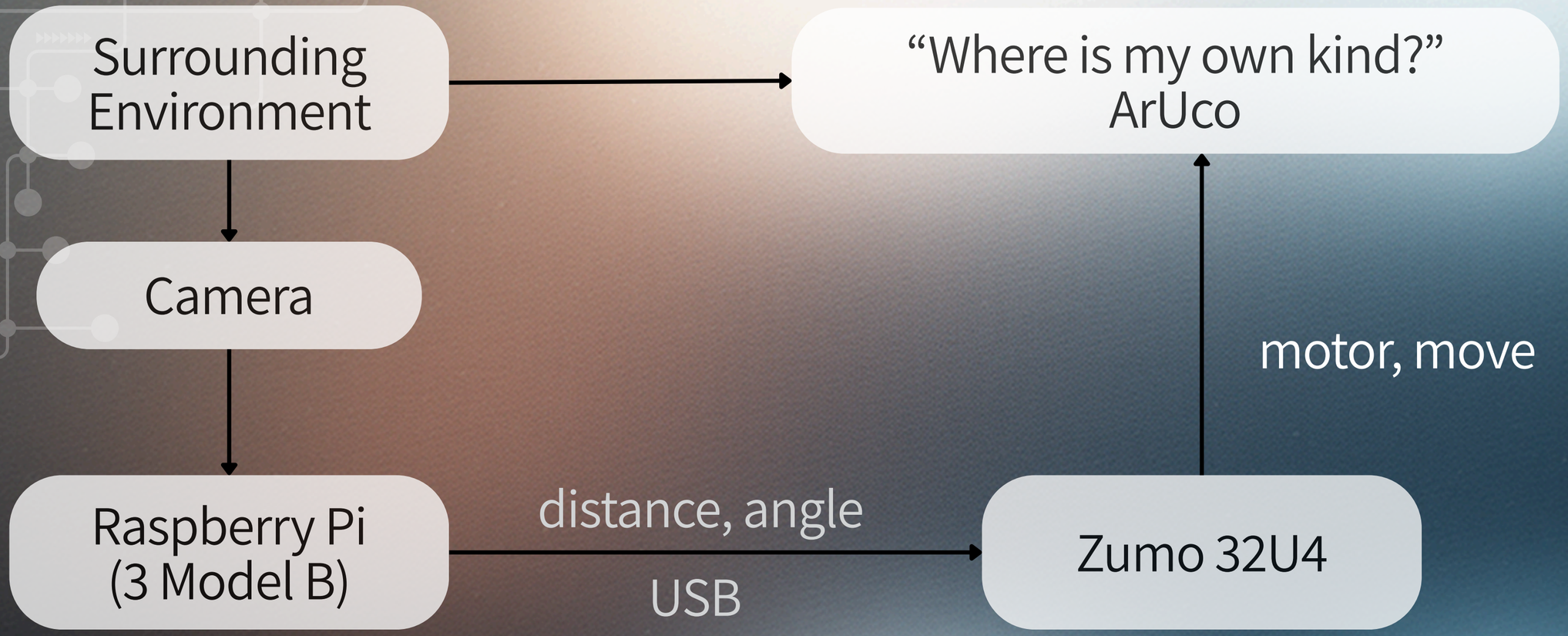
Raspberry Pi Camera Module 2

Solution 2

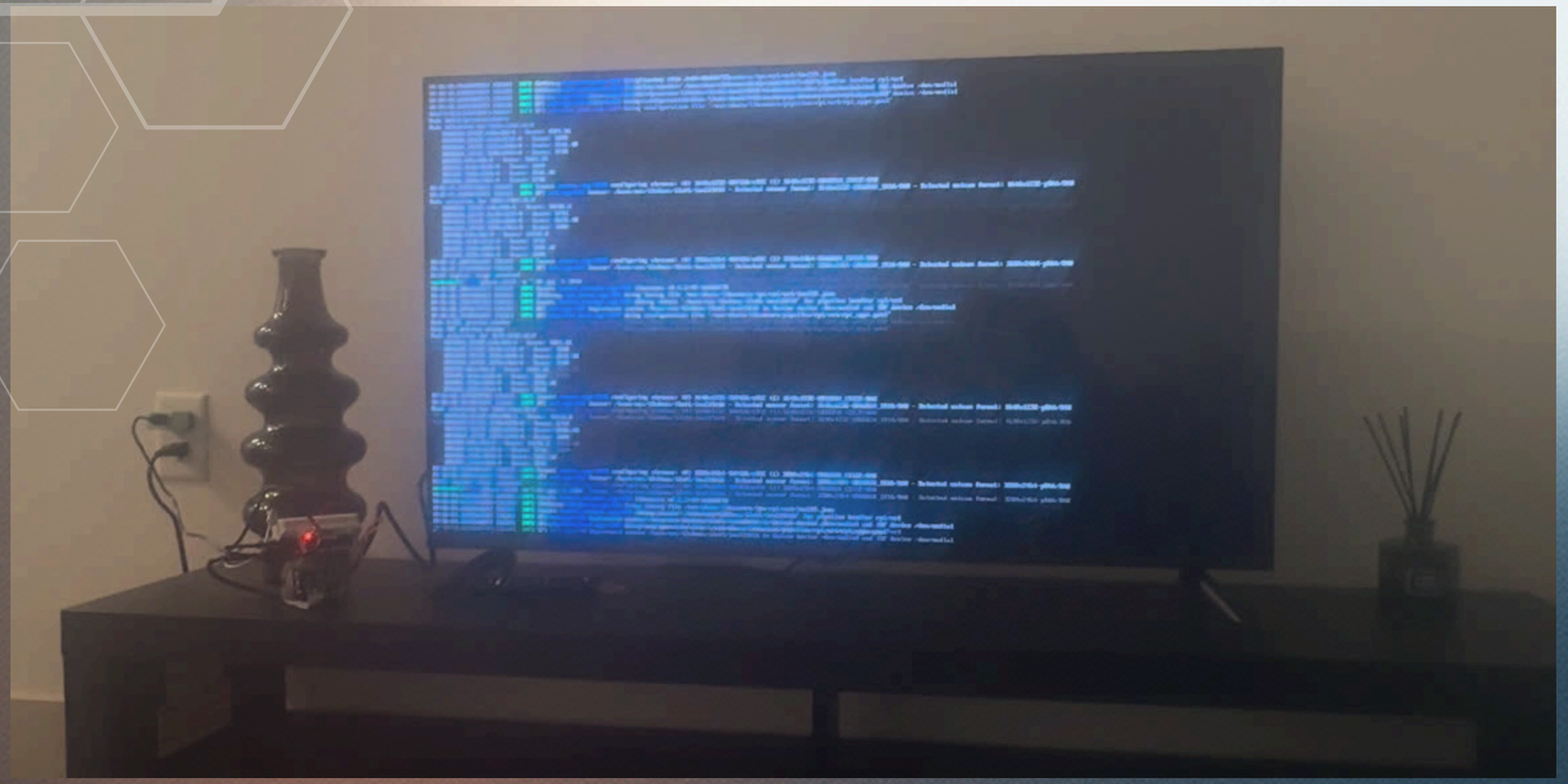
Use Rasberry Pi

Integrating a Raspberry Pi with the onboard camera allows the Zumo robot to visually recognize and track the vehicle ahead instead of relying only on infrared sensors. The camera continuously captures video frames, and the Raspberry Pi processes these images using computer vision algorithms (for testing now, ArUco) to identify their peer robot, then get the position and distance.

Solution 2

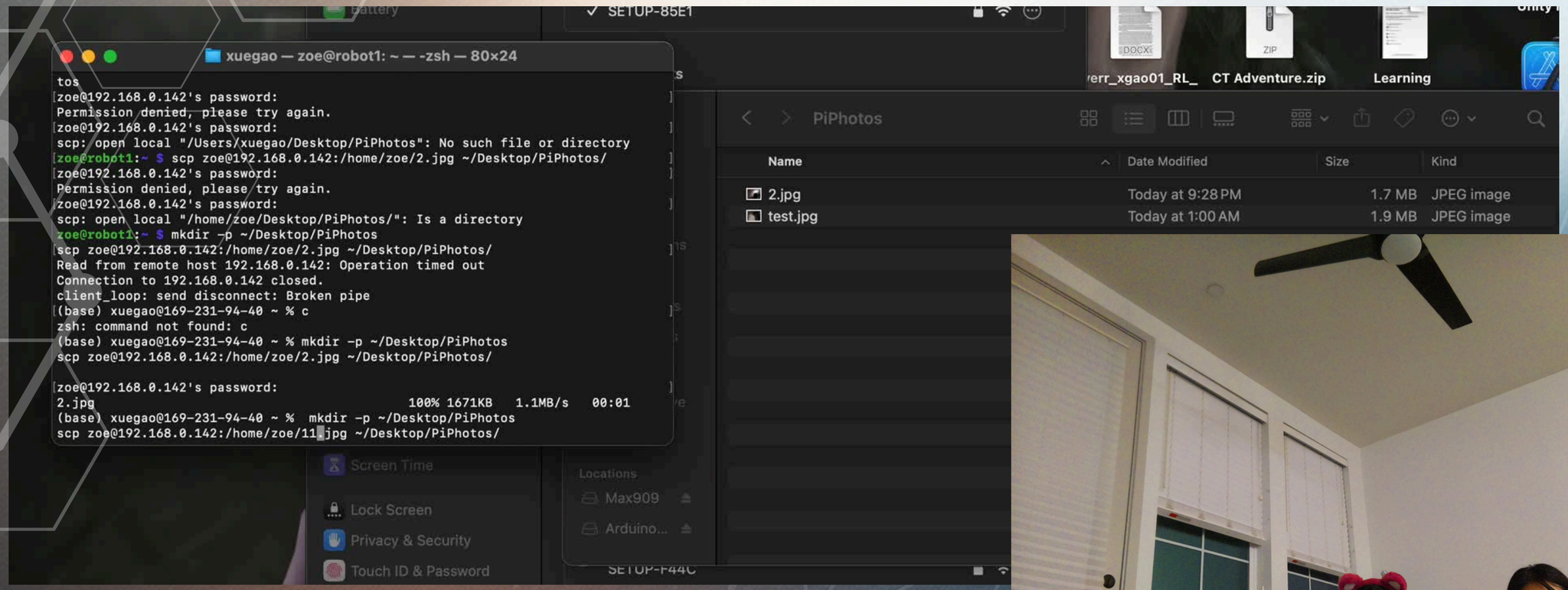

Surrounding Environment
“Where is my own kind?” ArUco
Camera
motor, move
Raspberry Pi (3 Model B)
distance, angle
USB
Zumo 32U4

Solution 2



We successfully accessed the Raspberry Pi system and repaired the attached camera module. After troubleshooting the hardware connection and software configuration, the camera is now functional.

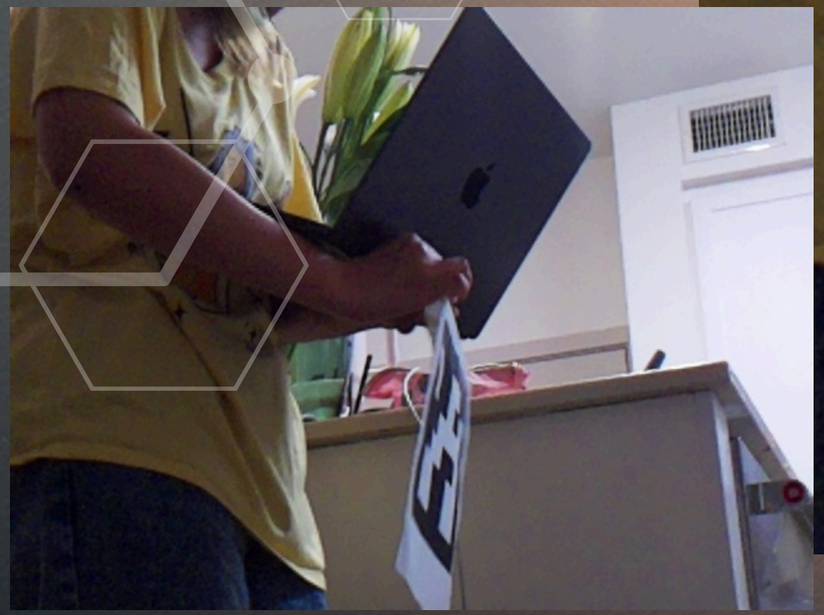
Solution 2



Demonstrated how to transfer photos taken to a laptop and save them.

Solution 2

- OpenCV ArUco
- GStreamer + libcamera
- cv2.aruco module to analyze
- compare pattern to dictionary



Solution 2

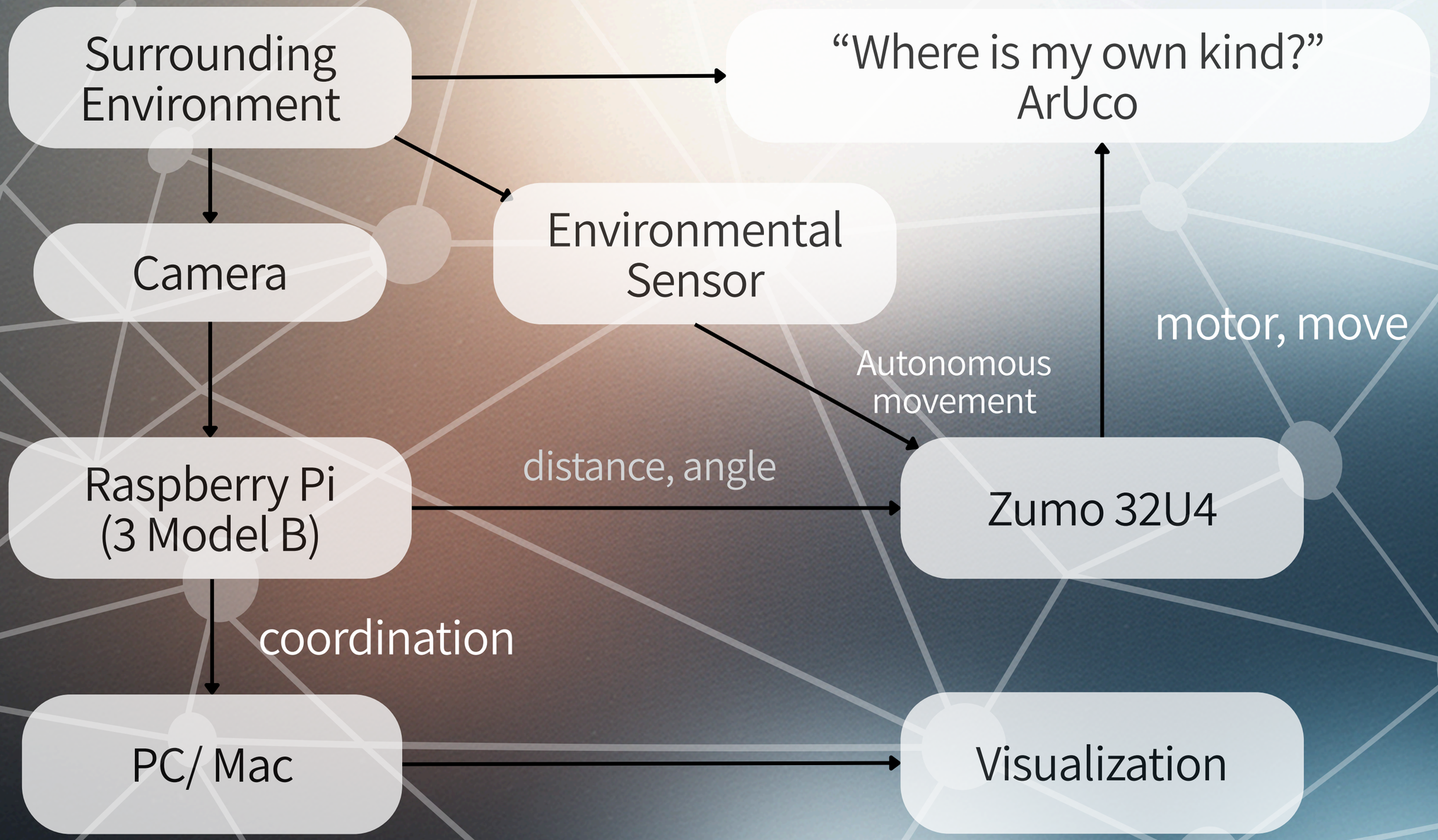
-What left to explore?

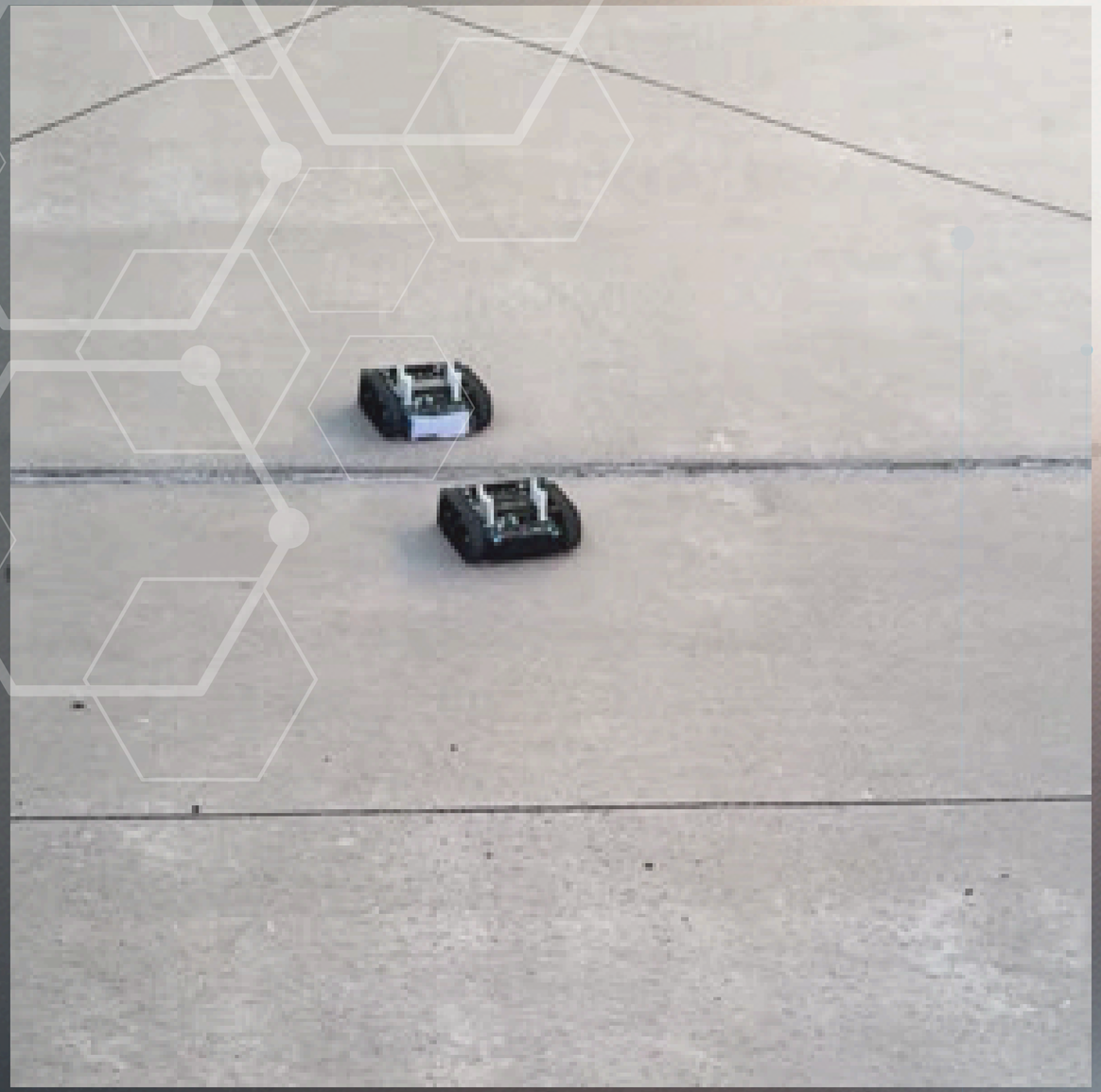
Link Raspberry Pi and Zumo

Visualize the movement of robots over time

What encourages autonomous movement of the robots?

Solution 2

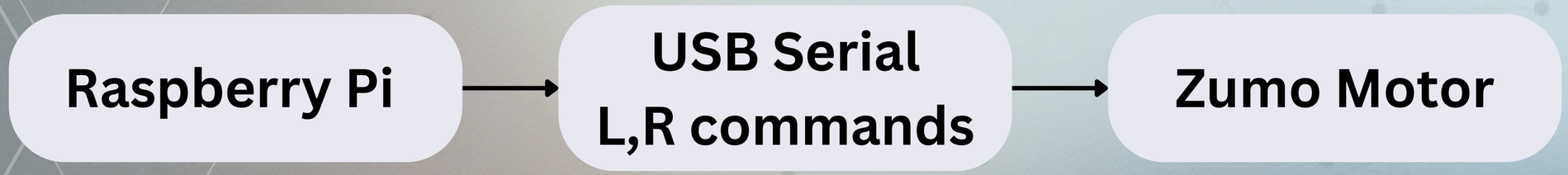

Surrounding Environment
“Where is my own kind?” ArUco
Camera
Environmental Sensor
motor, move
Autonomous movement
Raspberry Pi (3 Model B)
distance, angle
Zumo 32U4
coordination
PC/ Mac
Visualization



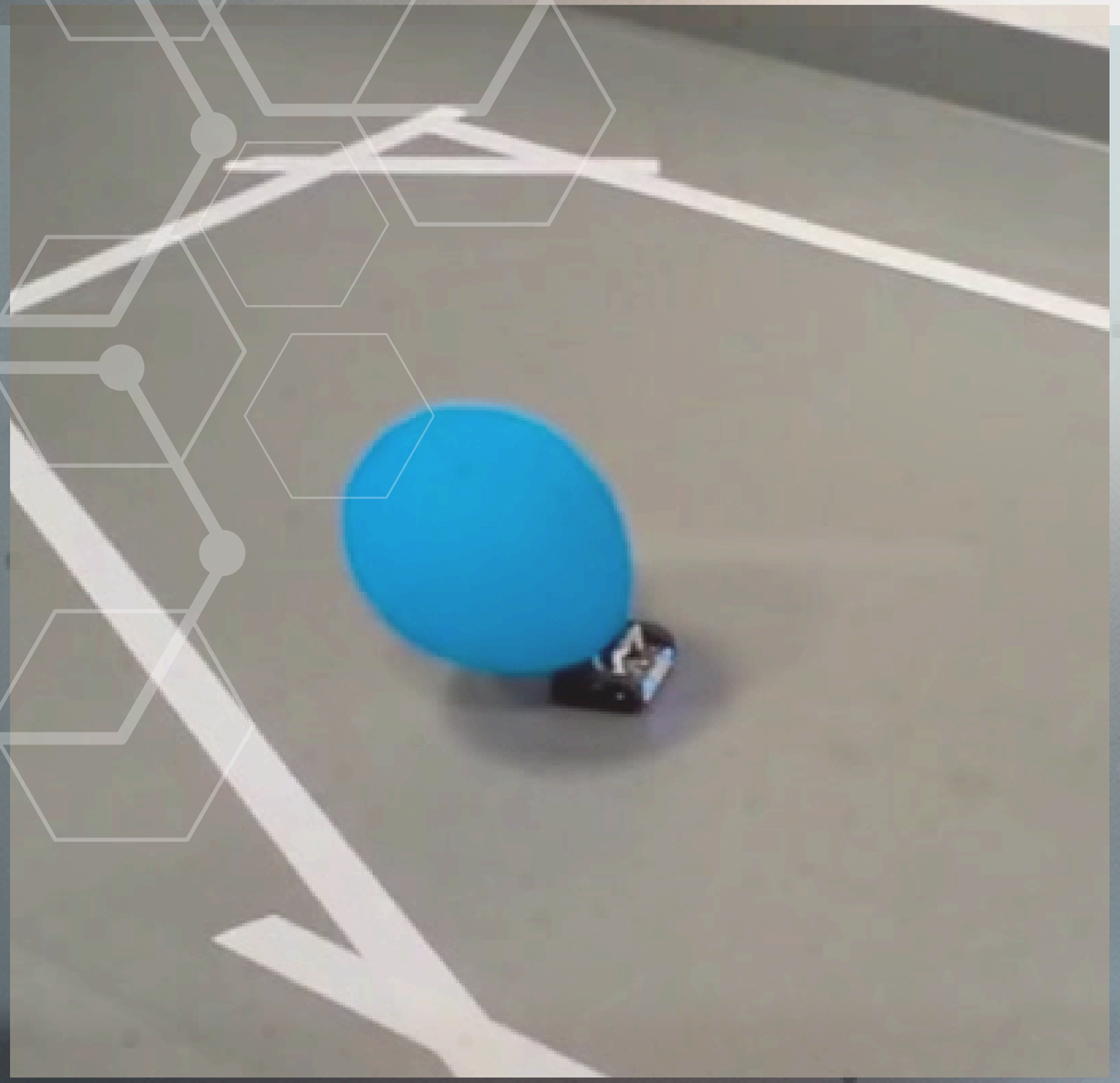
Testing : How the Rear ZUMO Follows the Front ZUMO

In this early experiment, we tested how the rear zumo responds when the front zumo moves according to a predefined pattern. The goal was to observe whether the follower zumo could accurately track and maintain distance using only visual recognition of the front zumo.

Follower:
Raspberry Pi Controll Zumo



- Pi sends motor commands to Zumo via USB serial
- motors.setSpeeds(L, R); Zumo reads and parses the string into left/right motor speeds.
- If no new command is received within 300 ms → robot stops.
- Pi handles vision & decision-making; Zumo handles motor execution.

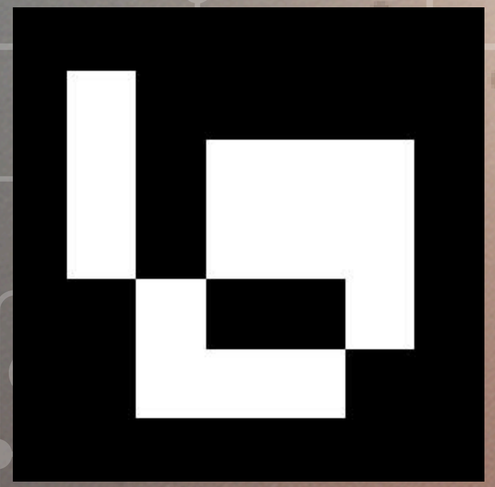


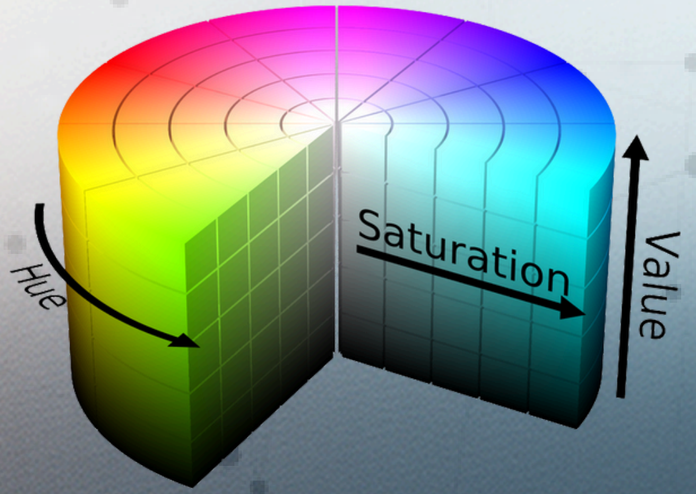
Leader: Line-Sensor Based Boundary Control

Using Line Sensors to Detect the Tape

- Zumo has 5 line sensors (L0–L4).
- Floor = bright (high values ~400–600)
- Tape = white (low values ~70–110)
- We use a threshold 300 to detect the tape.

Color Detection: Why We Switched & What We Fixed





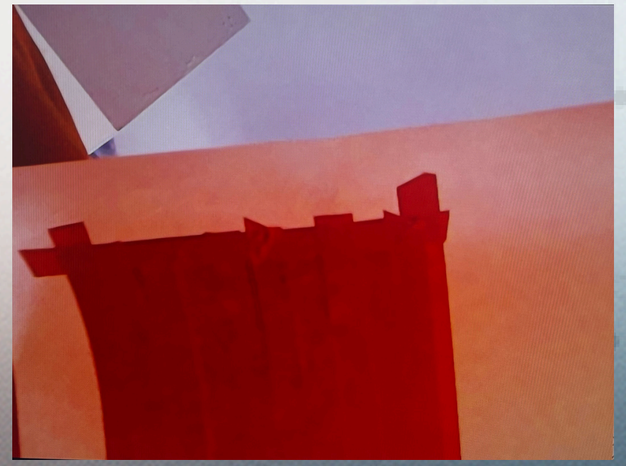
- At first, we used ArUco markers for detection.
- We wanted to use the marker size to estimate distance.
- In practice, ArUco was not stable: small, far, and easy to lose.
- We switched to color-based detection using HSV in OpenCV.
- We detect a blue region, and use its contour area as a proxy for distance.

Color Detection: Channel Correction

- Picamera2 outputs frames in RGB format.
- OpenCV expects images in BGR format for all color processing.
- To match OpenCV’s expected format, we reverse the channel order:

  `frame = frame[:, :, ::-1]`

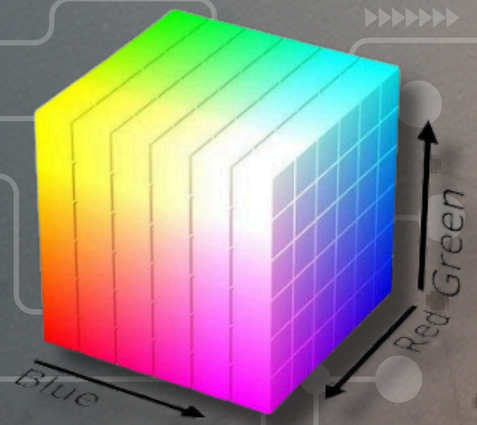
- This ensures the color detection use the correct color values.



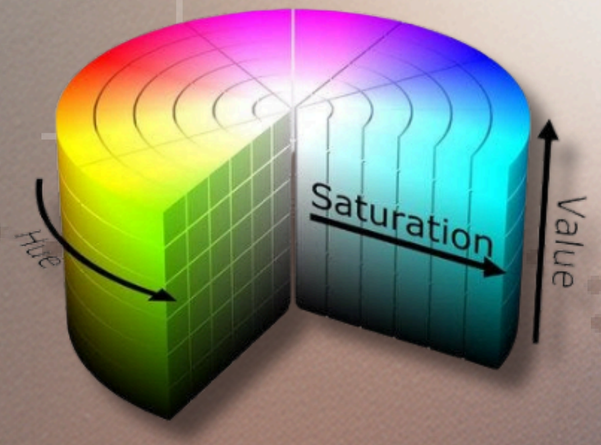
Before

After

Why HSV Instead of RGB



RGB



HSV

See also

cv::COLOR_YUV2RGB_NV12, cv::COLOR_YUV2RGBA_YUY2, cv::COLOR_BGR2YUV_YV12 and similar ones

RGB <-> HSV

In case of 8-bit and 16-bit images, R, G, and B are converted to the floating-point format and scaled to fit the 0 to 1 range.

$$V \leftarrow max(R, G, B)$$

$$S \leftarrow \begin{cases} \frac{V - min(R,G,B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$H \leftarrow \begin{cases} 60(G - B)/(V - min(R, G, B)) & \text{if } V = R \\ 120 + 60(B - R)/(V - min(R, G, B)) & \text{if } V = G \\ 240 + 60(R - G)/(V - min(R, G, B)) & \text{if } V = B \\ 0 & \text{if } R = G = B \end{cases}$$

If $H < 0$ then $H \leftarrow H + 360$. On output $0 \leq V \leq 1, 0 \leq S \leq 1, 0 \leq H \leq 360$.

The values are then converted to the destination data type:

- 8-bit images: $V \leftarrow 255\,V, S \leftarrow 255 S, H \leftarrow H/2$(to fit to 0 to 255)
- 16-bit images: (currently not supported) $V \leftarrow 65535\,V, S \leftarrow 65535 S, H \leftarrow H$
- 32-bit images: H, S, and V are left as is

- Hue is more stable under changing light than RGB.
- HSV separates color (H) from brightness (V), which reduces false detection.
- Final HSV Color Range: lower_blue = np.array([100, 120, 60])
  upper_blue = np.array([140, 255, 255])

Improving Detection Robustness

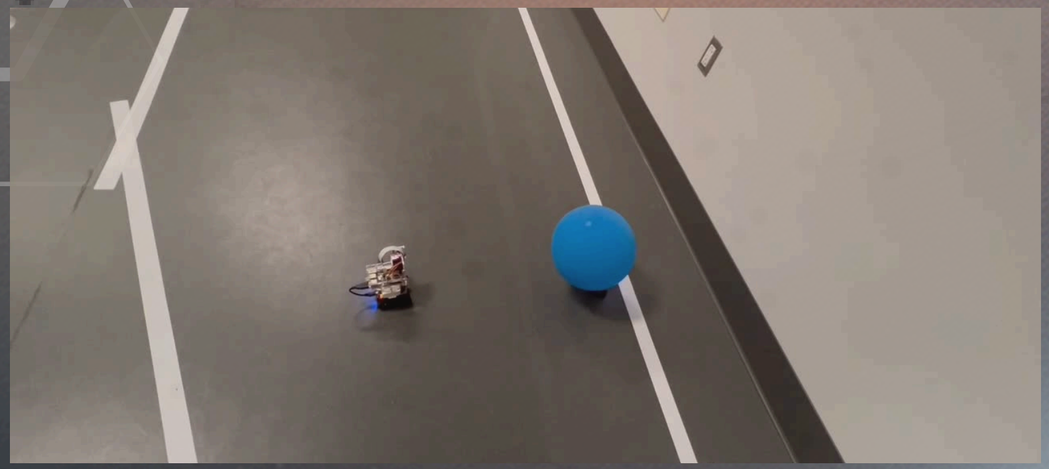
```
MIN_AREA = 2000

mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)   # remove small noise
mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)  # fill small gaps
...
if area < MIN_AREA:
    return None, 0.0, area
```

- Open + Close operations remove small noise and fill minor holes.
- MIN_AREA filtering rejects tiny or distant blue spots.
- This ensures the robot is not affected by glare, reflections, or random blue pixels.

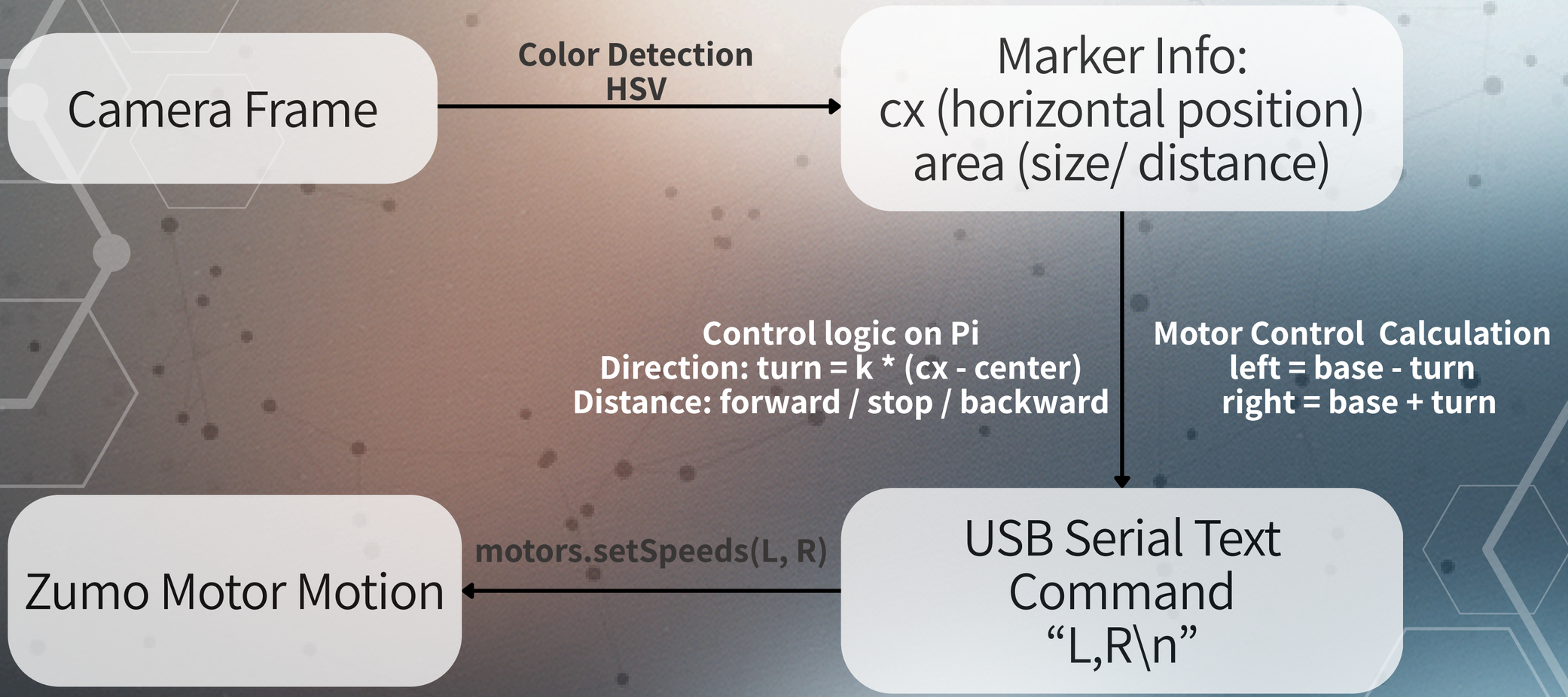
Improving Detection Robustness

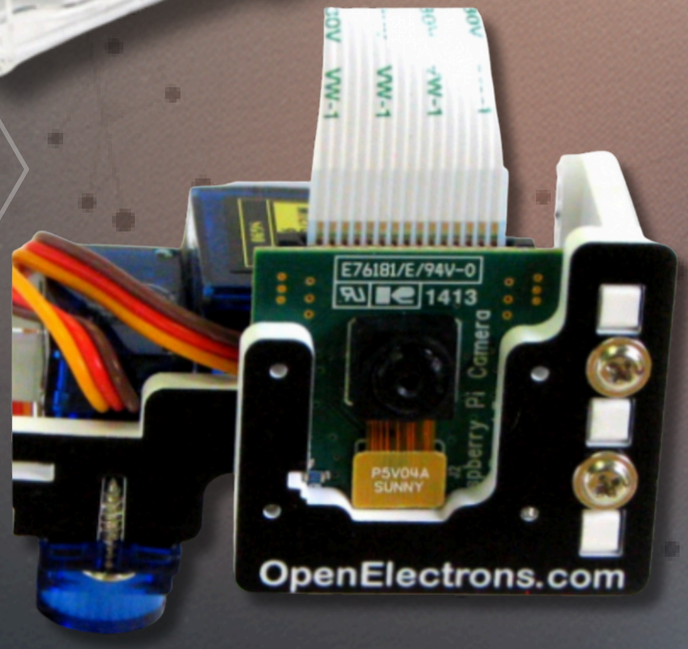
- Proportional steering based on cx offset
  - cx = The x-coordinate of the blue marker’s centroid in the camera image.
  - cx < center → turn left
  - cx > center → turn right
  - K- Proportional Gain to control turn speed



```
error = cx - image_center
turn = k * error
```

Raspberry Pi - Zumo Pipeline





Pi-Pan Servo System Overview

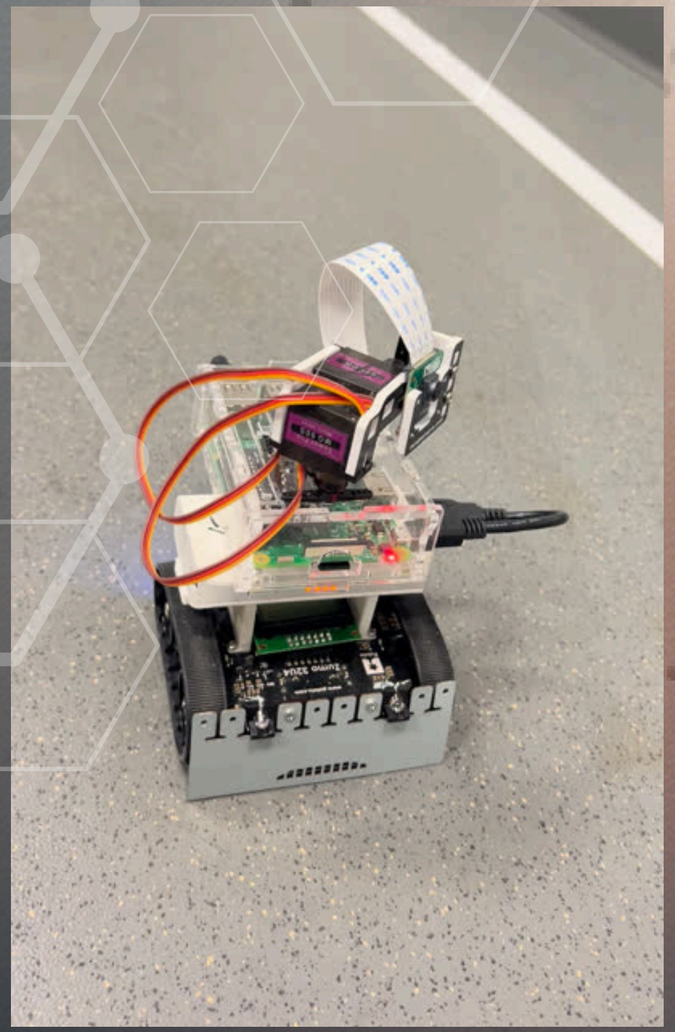
Servo Hardware

- Pi-Pan uses two TowerPro SG90 micro servos.
- Rotation: ~180°
- Standard connector: Brown=GND, Red=+5V, Orange=Signal.

https://www.mindsensors.com/content/24-pi-pan-readme

```
cd ~/pi-pan
sudo ./servod
cd ..
```

Pi-Pan Servo System Overview



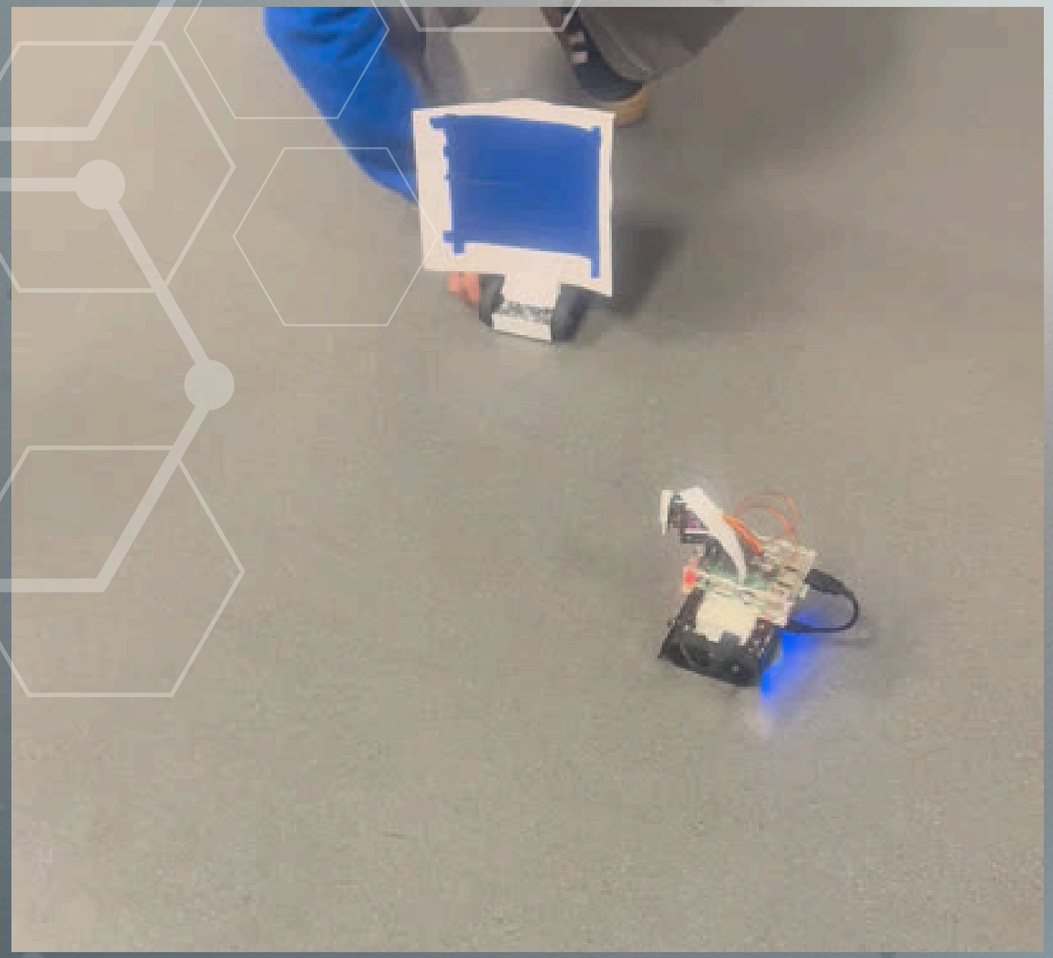
PWM Control Signal

- ~0.5 ms pulse → ~0°
- ~1.5 ms pulse → ~90° (center)
- ~2.4 ms pulse → ~180°

Servo Channels

- Channel 5 → Pan servo
- Channel 6 → Tilt servo
- In our project, we only use Pan (Channel 5).

Why Our First Servo-Based Test Failed

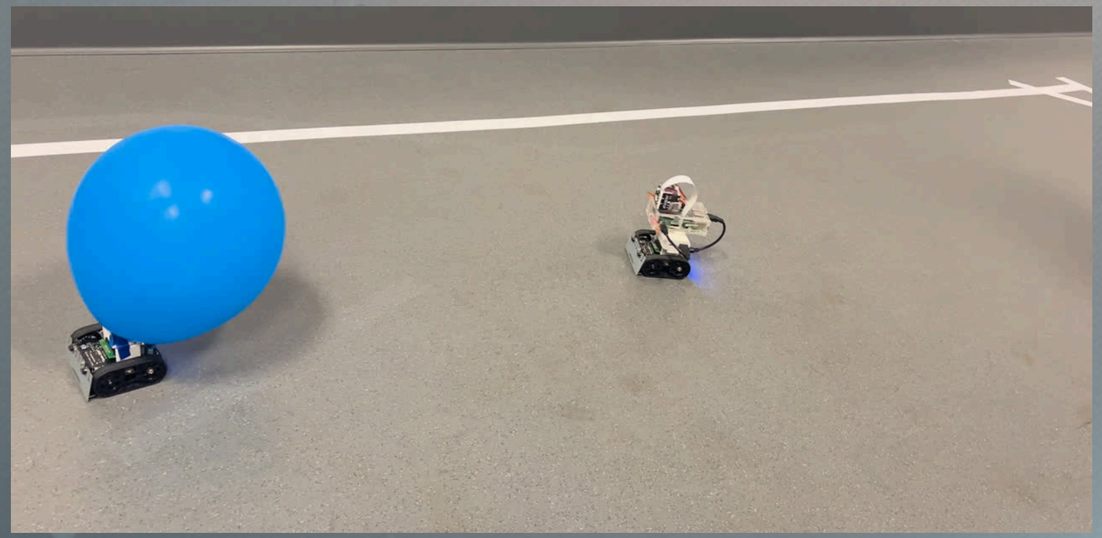


Core Summary: Servo-based searching caused infinite spinning

- Servo was used for both searching and aligning
- Searching: servo sweeps left↔right, camera follows
- Alignment: Zumo turns based on servo angle
- Problem: servo angle ≠ robot center → inconsistent orientation
- Robot keeps turning even when already aligned → spinning loop
- Servo introduces latency, making feedback unstable

Body Rotation Search Strategy

```
if mode == "search":
    # Spin in place
    left_cmd  = SEARCH_TURN_SPEED *
MOTOR_LEFT_SIGN
    right_cmd = -SEARCH_TURN_SPEED *
MOTOR_RIGHT_SIGN
    set_motors(left_cmd, right_cmd, tag="[SRCH]")
```



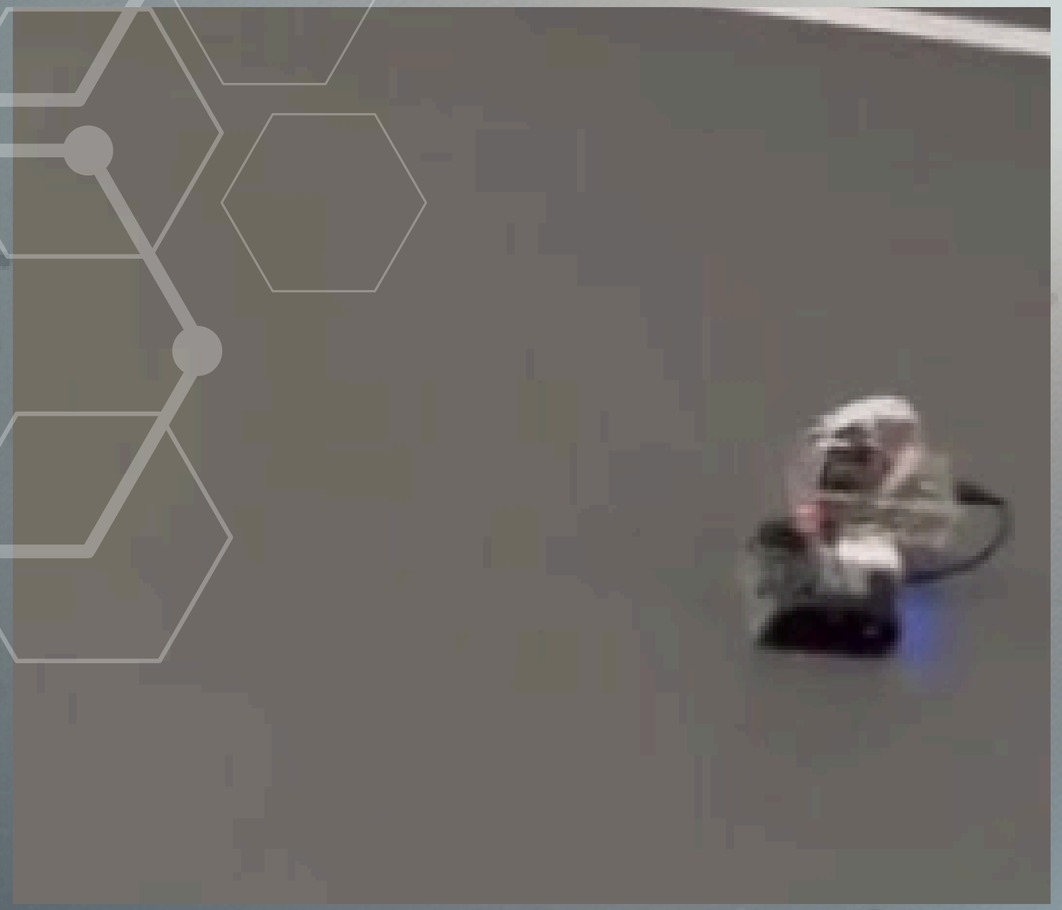
What we do now

- When the target is lost:
- → Zumo rotates its body in place to search.
- No more head scanning or servo movement.
- Robot stops rotating as soon as the blue region is detected again.

Advantages

- Much simpler logic
- More reliable reacquisition
- No alignment mismatch
- Stable and predictable tracking behavior

Rotation Speed Experiments



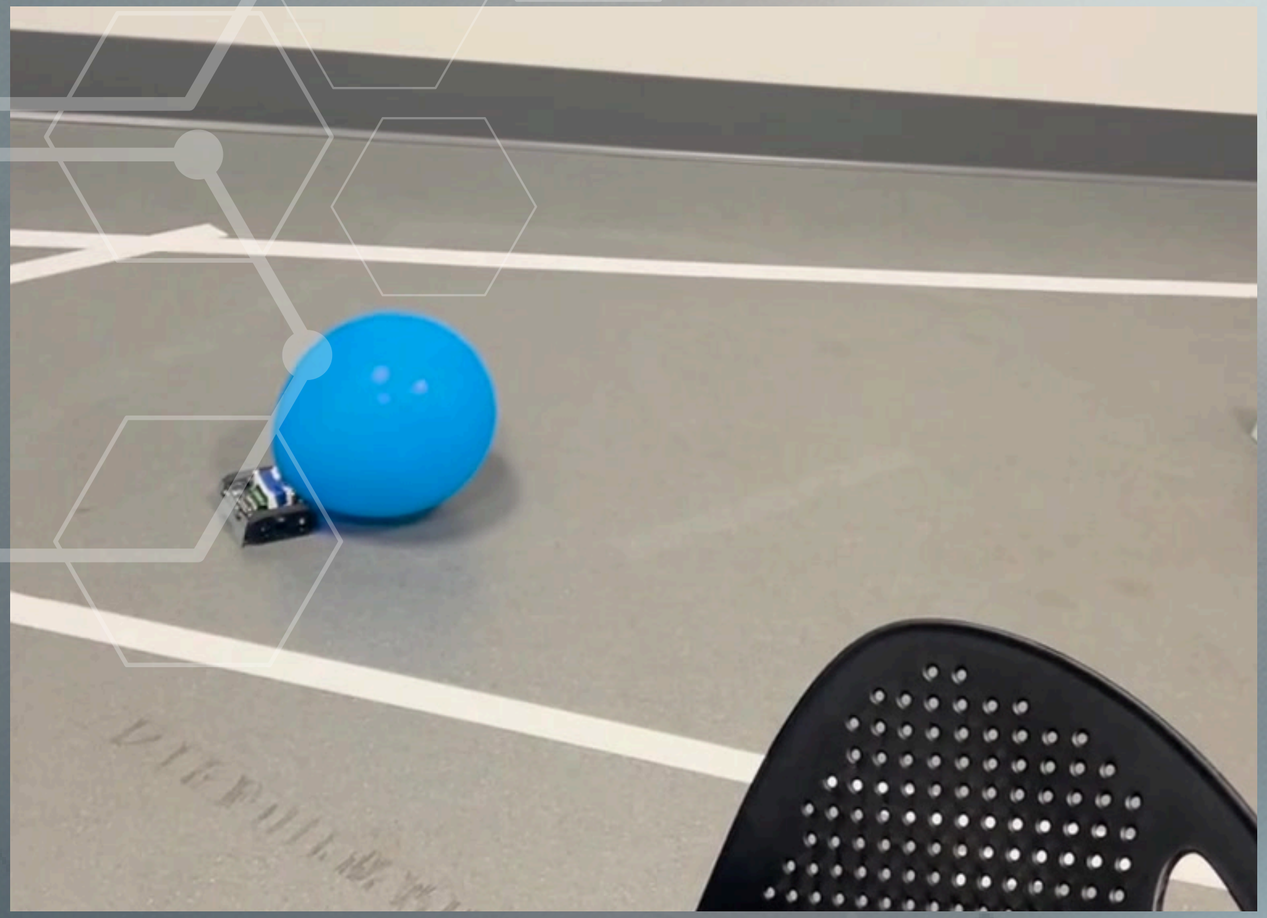
What we tested

- Rotation speed < 70 → motors stall (robot cannot rotate)
- Rotation too fast → camera misses frames, cannot reacquire target
- Tried:
  - Slow rotation
  - Rotate → pause → detect
  - Hybrid slow + pause method

Final choice

- Constant rotation at speed 70–90
- Smooth enough for camera
- Fast enough to avoid stalling
- Most reliable reacquisition behavior

Minimum Distance Behavior



Problem

- If the follower gets too close, the blue region becomes too large → detection becomes unstable.
- Stopping only can cause direction flipping and tracking errors.

Tests

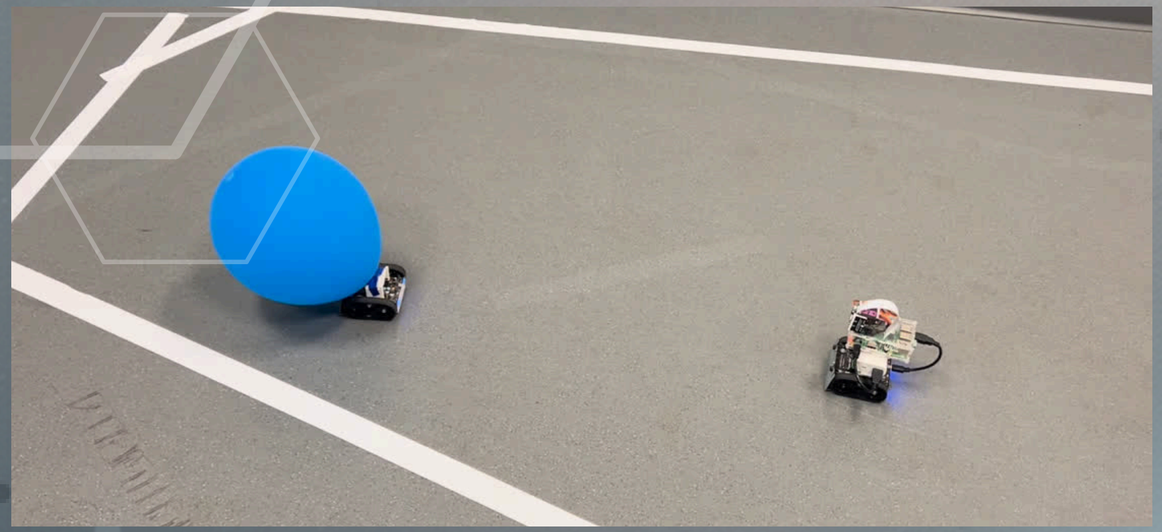
- Stop only
- Reverse slightly
- Reverse + slow turn

Final Choice

- Always reverse when distance < threshold
- Keeps target size stable in the camera
- Avoids wrong-direction flips and improves tracking consistency

Tracking Logic (Displayed in Presentation)

```
# ---- State Machine ----
if mode == "search":
    if cx is not None:
        mode = "track"
        lost_frames = 0
elif mode == "track":
    if cx is not None:
        lost_frames = 0
    else:
        lost_frames += 1
        if lost_frames >= LOST_MAX_FRAMES:
            mode = "search"
```



Goal

- Implement a robust, stateful controller that can:
- Search for the blue marker when it is lost
- Move forward when the marker is visible
- Recover gracefully when detection fails

State Machine

- Two modes: search and track
- Search mode
  - Robot spins in place at SEARCH_TURN_SPEED
  - Uses HSV + morphology + MIN_AREA to detect a large enough blue blob
- Track mode
  - When marker is visible → go straight forward
  - (TRACK_FORWARD_SPEED)
  - If marker is missing:
    - Increase lost_frames
    - After STOP_AFTER_LOST frames → temporarily stop
    - After LOST_MAX_FRAMES frames → switch back to search

Motor Sign Strategy

```
MOTOR_LEFT_SIGN = -1
MOTOR_RIGHT_SIGN = -1
```

```
MOTOR_LEFT_SIGN = 1
MOTOR_RIGHT_SIGN = 1
```

Different Zumo units have different physical motor orientations, so we abstract the motor direction into SIGN variables. Adjusting 1/–1 lets us correct the direction without changing the control algorithm

Test Summary

Category	Detection	Search Method	Search Pattern	Motion & Turning	Distance Control	Target Loss Handling	Safety & Robustness	Debugging
Options Tested	ArUco marker ↔ HSV color	Servo scan ↔ body rotation	Rotate–pause ↔ continuous	Hard turn ↔ cx-basedNo cap ↔ capped turn	Stop ↔ reverseSingle threshold ↔ bands	Immediate search ↔ frame counterReuse last cx ↔ re-search	No protection ↔ timeoutRaw speed ↔ deadzone	No record ↔ snapshots
Issues Found	ArUco unstable, highly depend on shape and size	Servo misalignment, spinning	Rotate–pause cause continuous target missing. If rotate speed is too slow, motor can't rotate; if rotate speed is too high, cause detection failure	Oscillation;sudden sharp turns	Target fills frame → cx failure;oscillation near boundary	Too sensitive;wrong turning after loss	Runaway risk;motors fail to move at low speed	Hard to analyze failures
Final Choice	HSV color detection with size thresold and noise handling	Zumo rotates in place	Continuous rotate with a speed limited between 70 - 90	Proportional steering;max turn speed limit	Reverse when too close;near / good / far bands	Lost-frame counter;prefer re-search	Serial timeout stop;deadzone compensation	Photos on state change

Remaining Issues

- Leading car must move very slowly
  - To ensure the follower can reacquire after losing the target,
  - the front car’s speed has to be extremely low.
- Follower may fail after sudden turns
  - If the front car turns too sharply,
  - the follower briefly loses the blue marker and sometimes cannot re-align.
- Motor direction inconsistency
  - Physical wiring and orientation differences require adjusting
  - MOTOR_LEFT_SIGN / MOTOR_RIGHT_SIGN,
  - and inconsistent sign settings can cause reversed turning behavior.

Xue Gao

Zixuan Zhang

MAT 265

2025