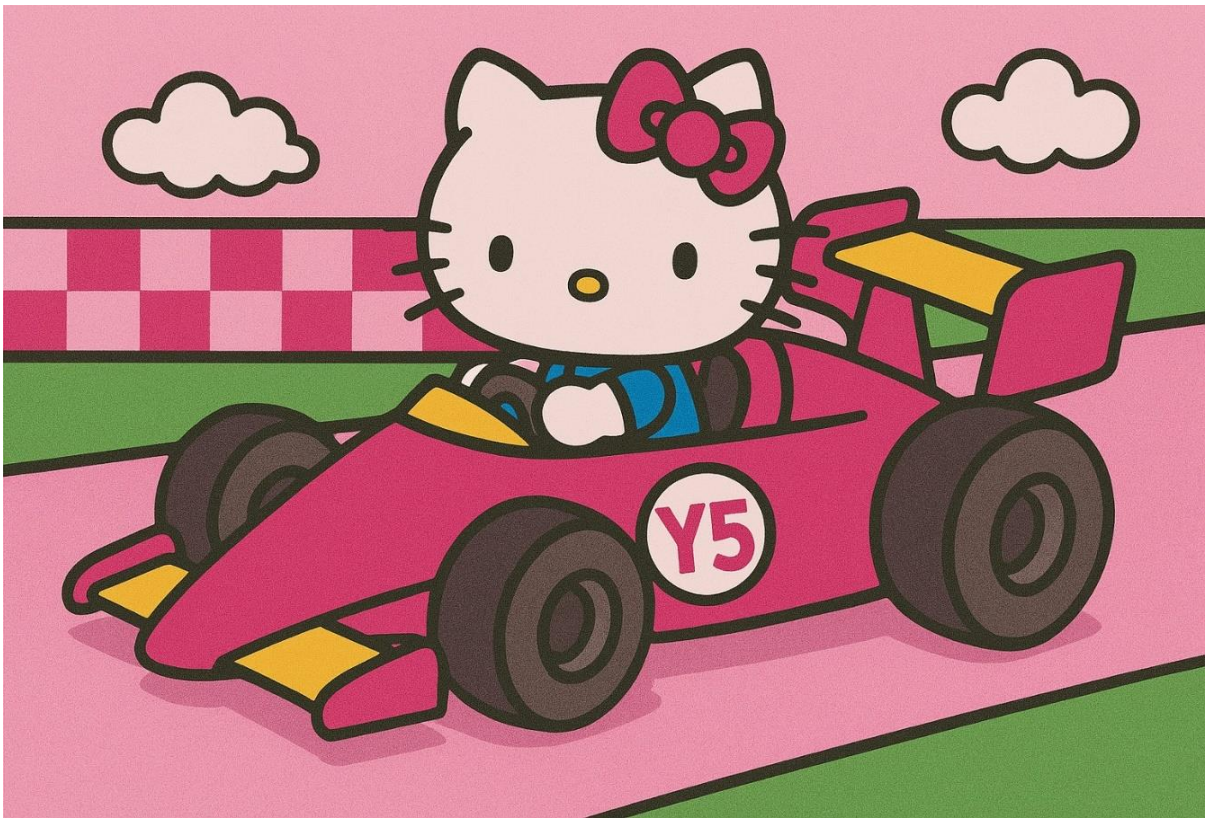
Y05 Final report



By

Abbie McQuaid
Grace Gallagher
Abdulrahman Elsibaei
Leah Dunne

Introduction

This report outlines the development and testing process of our autonomous buggy project. The objective of the project was to design, build, and program a self-driving buggy capable of line-following, object orientation, and object following, tested by a series of bronze, silver and gold performance challenges. Our approach combined collaborative teamwork, iterative testing and strategic debugging to gradually improve the buggy's capabilities and reliability. Our team worked collaboratively, assigning clear roles and constantly communicating. We successfully completed the bronze challenge and performed well in the silver challenge, particularly in object detection and following.

However, technical setbacks- such as power instability, sensor issues and calibration problems, limited our progress to complete the gold challenge. A key issue was a drop in battery voltage during the silver demo, which disrupted the PID tuning. The report reflects on our design process, team dynamics, technical challenges, and ethical considerations of autonomous systems, offering insights into both our successes and areas for future improvements,

Part A – Self-assessment

Team performance assessment

Strengths:

- Clear roles within the group allowed us to complete tasks on time and to the best of our ability.
- Frequent testing of the buggy allowed us to properly identify issues with our buggy and fix them before the challenges.
- We achieved accurate object detection & object following in both bronze and silver challenges.

Weaknesses:

- Our power supply was unstable when testing for silver our batteries died, and we had to replace them with new ones from another group. The new batterie's were more powerful and resulted in us having to change the k-value in our code, this meant we didn't pass silver, the first time around. This weakness of poor planning cost us the gold challenge.
- Although our navigation of the path was pretty good the buggy did have a slight jerky motion and sometimes veered off the track when turning corners. This jerky motion made the buggy less stable in the silver challenge.
- In the first weeks of the module, we had a problem with calibrating the speed of the wheels. This resulted in a lot of lost time as we initially assumed it was a problem with our code instead of a problem with the wheels themselves. This mindset of the problem only being the software rather than the hardware meant we waste a lot of time.

If our team achieved silver on time, identify 3 changes to our approach that would have allowed us to achieve gold if given 2 more weeks to work on our design?

On the day of our Silver Challenge demo, the batteries in our buggy had to be replaced shortly before testing as they were dead, and we didn't have enough time to charge them. The new batteries had a higher voltage than the drained ones used during development and testing, which increased the responsiveness of the motors. As a result, the previously tuned K-value for our PID speed controller was no longer suitable for our buggy. When we tested the buggy, it was

too fast as it was not turning corners properly and was veering off the track, as it was picking up too much speed along the straight parts of the track that when it came to turn the corners it was already travelling too fast and could not turn the corner successfully. This caused the buggy to behave erratically, with overshooting and instability in speed control. To fix this, we had to quickly alter the K-value in the code to regain stable performance. Unfortunately, we couldn't find the correct k-value in time to demo the buggy, and we missed out on gold. I think given 2 more weeks to demo for gold we could have tuned the k-values perfectly to allow the buggy to perform perfectly on the track, as all other elements were working accordingly. We also would have learned from our mistakes and ensured our batteries were fully charged and able to be used in the silver demo, I think if we as a group had 2 more weeks, we would have completed the silver challenge in time to try gold.

If given 2 more weeks we could have also implemented more testing of the buggy as a group. The buggy would definitely have benefited from more testing as this meant we could have identified more problems with the buggy's hardware and software.

In conclusion, given an extra 2 weeks to complete the silver challenge would have ensured our buggy was not rushed in any way and that the buggy was fully ready. It would have also given us time to implement proper battery demo preparations and protocols as that was our main downfall when demoing for silver initially.

Ethical issues that may arise in the project.

1. Privacy and data collection

- Since the buggy uses sensors to collect data and eventually uses a “Huskey lens AI camera” these sensors and cameras can unintentionally collect data like faces, voices and locations.
- To avoid these issues a group could avoid unnecessary data logging.

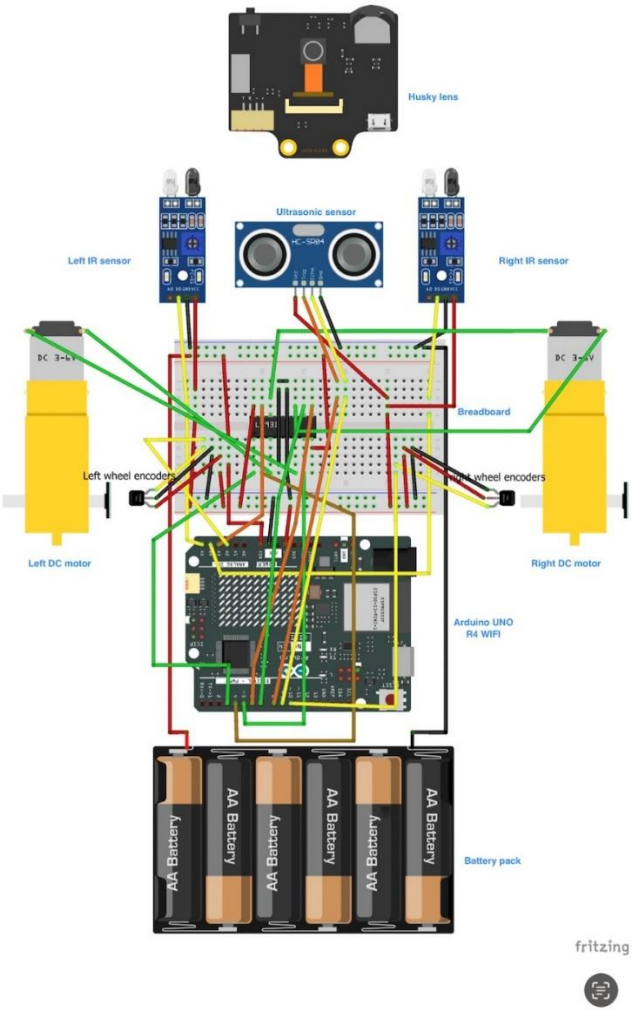
2. Potential for Misuse

- As the buggy is an autonomous system, the buggy could be repurposed for unintended applications, such as surveillance or tracking, which could lead to ethical concerns regarding misuse. For example, if the system was modified to follow individuals or navigate restricted areas, it could pose risks to privacy and security.
- To combat this, it is important to establish clear usage guidelines to prevent potential misuse and ensure transparency in its design and intended applications

3. Safety and Reliability

- Autonomous systems must be designed with safety in mind to prevent accidents, particularly in environments where they interact with humans or other objects. While the small size of the buggy makes it unlikely to cause physical harm, its components are valuable and require careful handling to avoid damage. To ensure safe operation, the buggy's motion, speed, and responsiveness were controlled to prevent erratic behavior.

Part B – Hardware design



Part B – Hardware design

Power supply

The circuit is powered by a battery pack connected to the breadboard's power rails, with the positive rail receiving voltage (5v) and the negative rail serving as ground. The Arduino Uno R4 Wi-Fi is powered through the battery and also supplies 5V to components of the circuit such as the sensors and the H-bridge.

Buggy design

The H-bridge acts as a junction between the battery and the DC motors, allowing control of motor speed and direction using pulse width modulation signals from the Arduino. Each DC motor is connected to the H-bridge with two wires, while the wheel encoders are powered through the breadboard and send rotational feedback and data to the Arduino. The ultrasonic sensor is powered by the Arduino and communicates via pins. Infrared sensors are also powered by the Arduino and send signals to digital input pins for line detection.

Circuit elements and their connections

Battery pack	The positive terminal is connected to one the positive rails on the breadboard and the negative terminal is connected to the negative rail which is the ground
Power rails of the breadboard	The positive rail is connected to the voltage input from the battery pack and the other positive rail is connected to the Arduino 5v voltage output. The negative rail on both sides of the breadboard is connected to each other which connects the Arduino and the battery ground.
Arduino Uno R4 Wi-Fi	The Arduino processes the data from the sensors as well as controlling the motors and acts as a server for wireless communication with the laptop. It is powered through the voltage input pin which is connected to the battery pack , it essentially acts as the brain in our buggy
H-Bridge	From figure 1 the H-bridge logic circuitry Vcc 1 is powered by the Arduino 5v output

	pin and Vcc 2 is the only pin that is directly powered by the battery pack and the H-bridge acts as the middleman between the battery pack and the dc motors using the enable pin 1 & 2 you can control the speed and direction of the DC motors using pulse width modulation (PWM). The input & output pins 1 & 2 are responsible for the left motor and 3 & 4 are responsible for the right motor.
DC motors	Both left and right motors have 2 wires connecting them to the H-bridge. The DC motors are operated with pulse with modulation. Control signals are sent via the Arduino to the H-bridge, the H-bridge controls the flow of voltage from the batteries to the DC motors to allow them to turn.
Left & Right wheel encoders	They are both connected to the positive and negative breadboard terminals Each wheel encoder (left & right) has one single wire connecting them to the Arduino digital/analog pin which are set to receive an input from the encoders which send a low to high signal 4 times per revolution of the wheels
Ultrasonic sensor	The voltage input comes from the Arduino (5V) It is then connected to ground Trig and echo are connected to the Arduino to send and receive signals
Left & right infrared sensors	The voltage is supplied via the Arduino with wires connecting both sensors to the 5V breadboard terminal There are wires connecting the sensors to the Arduino digital input pins for the signal output of the sensors

Part C – software design

Pseudo code Arduino:

INITIALIZE global variables:

- referenceSpeed = 0.0
- objectFollowing = false
- distanceTravelled = 0
- okay = false (indicates whether the buggy should move or stop)
- WiFi connection details (ssid, pass, etc.)
- PID-related variables and constants (Kp, Ki, Kd for speed; objFollowing_Kp, etc. for object following)

SETUP FUNCTION:

1. Begin serial communication at 9600 baud.
2. Connect to WiFi using ssid and pass until successfully connected.
3. Start WiFi server on port 5200.
4. Initialize the LED matrix display and load a default frame/pattern.
5. Set up infrared sensors and encoders.
6. Configure PID controllers:
   - For speed control:
     - Set the controller in AUTOMATIC mode.
     - Limit the output range (0 to 100).
   - For object following:
     - Set the controller in AUTOMATIC mode.
     - Limit the output range (0 to 100).

MAIN LOOP:

1. Update encoders to get the current speed and distance traveled.

   - currentSpeed = getMeanSpeed()

   - distanceTravelled = getMeanDistance()

2. Measure distance to an obstacle with ultrasonic sensor:

   - distance = ultrasonic.getDistance()

3. Determine if we should be in object-following mode:

   - If distance is between 15 and 25, objectFollowing = true

     Else, objectFollowing = false

4. Check for client (Processing GUI) commands:

   - If a client is connected and has data:

     - Read the command until the '*' character, store in clientCommand

     - If command is "g" (go): set okay = true

       If command is "s" (stop): set okay = false

     - If command starts with 'v', the remainder of the command is the new referenceSpeed

5. If okay == true AND distance > 15:

   - Determine if we are in object-following or speed-control mode:

     - If objectFollowing == true:

       - Use object-following PID:

         - objFollowing_input = distance

         - Compute PID for objFollowing_output

         - motor.setSpeed(objFollowing_output)

- Else (normal speed control):
  - Use speed PID:
    - input = currentSpeed
    - setpoint = referenceSpeed
    - Compute PID for output
    - motor.setSpeed(output)
- Then apply line-following logic using left and right infrared sensors:
  - If both sensors see line (LEYE_Current == true, REYE_Current == true):
    motor.moveForward()
  - If left sensor is off-line, right sensor is on-line:
    motor.turnLeft()
  - If left sensor is on-line, right sensor is off-line:
    motor.turnRight()
  - Otherwise, motor.stop()

6. If okay == false OR distance <= 15:
   - Stop the motor

7. Send current data back to the client:
   - currentSpeed, distance, distanceTravelled, objectFollowing, etc.

8. Print diagnostic information to Serial (optional).

Pseudo code Processing:

DATA STRUCTURES:

referenceSpeed : float

currentSpeed   : float

distanceTravelled : float

objectDistance : float

objectTrackingMode : int   // 0 or 1

Kp, Ki, Kd : float   // optional PID parameters

prevReferenceSpeed : float

prevKp, prevKi, prevKd : float

client : NetworkClient   // manages TCP connection to Arduino

SETUP():

1. Initialize window of size (900x600)

2. Create UI elements:

   - Speed slider (range 0 – 25 cm/s)

   - Start button

   - Stop button

   - (Optional) Kp, Ki, Kd sliders

3. Connect to Arduino at IP <arduino_ip> on port 5200:

   client = new NetworkClient("192.168.xxx.xxx", 5200)

4. Initialize local variables and UI states:

   referenceSpeed = 10

   Kp = 0.59, Ki = 0.29, Kd = 0 // Or whatever default

   ...

MAIN LOOP (draw()):

1. Clear screen, draw background.

2. Read any available incoming data from Arduino:

   data = client.readStringUntil('*')

   if data is not null:

    parseTelemetry(data) // fill currentSpeed, objectDistance, distanceTravelled, objectTrackingMode

3. Update UI elements with current telemetry values:

   display currentSpeed, objectDistance, distanceTravelled, objectTrackingMode

4. Check if the user has interacted with sliders or buttons:

   - If Speed slider changed and referenceSpeed != prevReferenceSpeed:

     sendCommand("v" + referenceSpeed)

     prevReferenceSpeed = referenceSpeed

   - If Kp slider changed and Kp != prevKp:

     sendCommand("p" + Kp)

     prevKp = Kp

   - If Ki slider changed and Ki != prevKi:

     sendCommand("i" + Ki)

     prevKi = Ki

   - If Kd slider changed and Kd != prevKd:

     sendCommand("d" + Kd)

     prevKd = Kd

   - Start/Stop button event triggers:

     sendCommand("g") or sendCommand("s")

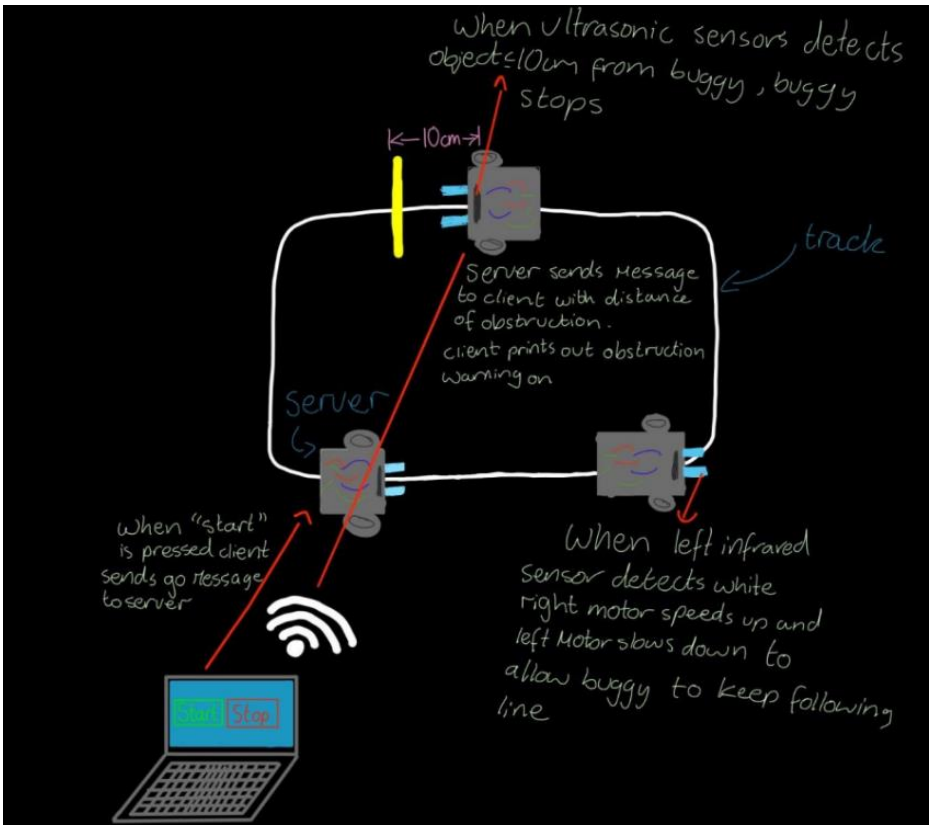
5. Refresh the display.

FUNCTION parseTelemetry(data):

```
 split data by ','
 currentSpeed = float(parts[0])
 objectDistance = float(parts[1])
 distanceTravelled = float(parts[2])
 objectTrackingMode = int(parts[3])

FUNCTION sendCommand(command):
 if client is active:
  client.write(command + "*")
 else:
  print("Not connected to Arduino")
```

Summary:



The buggy and the personal computer (PC) are connected to a mobile hotspot and the PC sends commands like “start” and “stop” or setting a reference speed and in response the buggy sends back updates like the current speed, distance to obstacle and total distance travelled to the PC. All this telemetry and settings can be viewed and modified from the graphical interface (GUI).

Once the start button is pressed the buggy begins to move and starts executing line following logic at a speed dictated by the reference speed or if a moving object is in front within a distance of 15 to 30 cm.

Part D

Challenges faced

1. One of the main technical challenges we faced when developing our buggy for the bronze challenge was achieving consistent straight-line motion as our left and right wheels needed to be calibrated to two different speeds for them to behave at the same speed and drive in a straight line on the track. This difference between the left and right motors caused the buggy to move in a snaking motion even if the line on the track was straight.

    - To resolve this, we had to do lots of calibration and testing. After multiple rounds of calibration and trial and error we managed to overcome the issue and successfully calibrated the wheel speeds. This meant the buggy was able to drive in a straight line on the track and meant the buggy was able to turn the corners more accurately.

2. The second challenge we faced was related to the reliability of the infrared sensors (IR) detecting the track. The buggy relied on 2 IR sensors on the front of the buggy to detect the track and in turn told the buggy to drive, stop or turn left or right. The buggy IR sensors often gave inconsistent readings and didn't detect the black and white parts of the track correctly. The unreliable readings from the sensors resulted in erratic movements of the buggy and complete deviation from the track.

    - We resolved this by understanding that the problem was caused by improper sensitivity of the IR sensors. To overcome this issue, we adjusted the sensitivity levels on the IR sensors until the buggy worked correctly on the track, following the track line properly.

3. A third significant challenge we faced was inconsistency in the buggy's power supply, which affected motor speed and overall performance. The buggy was powered by batteries, but as their charge level decreased, the voltage supplied to the motors varied, causing unpredictable changes in speed. This issue was particularly noticeable during extended testing sessions, where the buggy's behavior changed as battery levels dropped. The fluctuating power supply led to inconsistent performance, making it difficult to fine-tune motor speeds and sensor responses reliably. This was a critical issue because it meant that the buggy could behave differently from one test to another, even when running the same code.

    - To mitigate this issue, we implemented a more structured battery management strategy. Before each test, we ensured the batteries were fully charged, and we monitored voltage levels to determine when a battery change was needed.

4. The fourth challenge we faced was obtaining distance and speed readings from the encoders. The perimeter of a certain track is 3 meters, but our buggy would report about half the distance after completing a lap and speed readings were too volatile and due to the speed being based on the distance the values for speed were also halved

    - We suspected that one of the encoders weren't working and so in our code we have it that when the left and right encoder would output a high we have it increment a variable that stores the number times of the encoder outputs a high and we printed both the values to the serial monitor and we noticed that the left encoder wasn't returning anything and so we switch the pin to which it was connected to in the Arduino board and this resolved the issue. For the speed we averaged readings to reduce the impact of signal noise which produced sufficient stability in readings.

Conclusion

Our autonomous buggy project demonstrated the value of teamwork, iterative design, and hands-on testing in developing a functional self-driving system. While we achieved strong results in the bronze and silver challenges, technical limitations prevented us from advancing to the gold stage. These setbacks highlighted the importance of robust hardware calibration and power management in autonomous systems. Overall, the project provided valuable experience in system integration, problem-solving, and ethical considerations in robotics. The lessons learned will be essential for future engineering challenges and professional development.

Part E - Appendices

Arduino and processing code can be viewed in the submitted files