2E10 Final Report:

Group Z9.

Student Names and Numbers:

Guru Chappalle (23374967)

Joshua Searle (23373745)

Zac O’Connor (23374496)

Fintan K. Lalor (22335408)

13.04.2025

Table of Contents

Part A: **Self-Assessment**

Strengths

Effective teamwork

When a problem arose, we would meet up, discuss the issue, present ideas to solve said problem, and quickly agree upon the most effective solution. This helped by cutting down on each of us trying different ideas and instead focusing our time and energy on the same single solution. This proved to be efficient in saving us time and there was rarely a problem that was not solved by the agreed solution. For example, to calibrate the integral and differential coefficients, we met up and decided to use trial and error, noting the different coefficients we used and their effect on buggy acceleration and deceleration, when acceptable values were found (by reading the results from the Arduino) those coefficients were used and the problem of calibrating the coefficients was solved within 2 hours. It also helped greatly that everybody was present for all labs and meetings to discuss and work together.

Thorough analysis

All work was reviewed by another team member, which cut down on coding and wiring mistakes and also helped familiarize each other with new lines of code or new components on the breadboard and how they operate. We found that one is more likely to spot a mistake when reviewing past work rather than when writing new code or wiring new components. This helped with efficiency by solving issues close to when they arose, and not having to go back several weeks to solve a small error hidden in the coding files.

Creative and quick thinking

Ideas were thought of quickly and in numbers. We were rarely stumped on an issue without having any idea on how to solve it. From this, we were able to pick what we thought to be the best solution from a multitude of options. It is good to be able to view a problem from all angles, and having 4 individuals coming up with different ideas broadened each other's horizons and enabled us to improve each other's initial ideas. Outside of the box ideas helped massively as sometimes when presented with a problem, one would get tunnel vision and these creative ideas helped move us at a faster pace in fixing the issues faced.

Weaknesses

Testing

Looking back, we felt that we didn't test the buggy enough with our code, which led us to not passing the bronze challenge the first time around due to us not testing our code on the buggy until just before our demo. There was an issue with the wifi connection, and we were unable to do the demo successfully. This could have been avoided if we tested it earlier that day and found the issue earlier, solving it before our demo. We learned after this particular event and ensured to physically test new code on the buggy soon after it was written.

Initial Organisation Issues

Lack of clarity, although this was clearly part of the structure of the course. Having previously worked in modules whereby what was expected of us and how we were expected to do it. In this module we were given the target, some starting information, and then poof! Off you go. This freedom, while a welcome release in hindsight. Caused us to scramble around like headless chickens in the early stages of the term while we figured out exactly how we go about getting Maxine (our buggy) to complete the challenges before us, in a manner that was effective.

Even Distribution of Work

Throughout the project, as was raised at our interview, the required tasks were such that it was difficult to ensure a totally equal distribution of the workloads. Whilst carrying on from Bronze and towards silver and gold we at various instances tried to “reset” and redistribute work amongst ourselves. However these efforts were never wholly successful, between the nature of the work, our own proclivities and skill sets, and the messiness of college life between commuting and availability due to differing timetables and external commitments, the workload was not even but every effort was made to support each other as we went about this project.

Improved Approach to Gold

It would be fair to say that the organisational style used throughout the project could be described as Ad Hoc. The following headings discuss how we would suggest improvements to ensure a mitigation of the weaknesses discussed above.

Code Management & Proper Programming Practice

In the initial stages, sharing of code was rather primitive, using documents over Google Drive to share code across the group. While during the silver challenge we evolved to using Github the time to adjust was slow, and painful. It was only really in preparation for Gold we came to understand what we were working with. To that end, a more thorough introduction to Github would have been advisable. We were also over reliant on the header (.h) files. Whereas normally one has header files to introduce related functions for use in the main (.ino) file to implement these functions. We were over-reliant on the header files and management to handle both definition and and implementation within the header files contrary to convention.

Following convention both in writing and storing the code, would have improved Maxine's efficiency and made life much easier for all involved.

Scheduling

Despite best intentions our ventures to meet and work weekly collectively outside of the lab sessions rarely happened. Only once did the whole group manage to meet on our scheduled Thursdays, with one or more of us being called away to other things due to the clash of schedules. Perhaps the use of a management software or even a spreadsheet as we had in the initial reports, would have been much more effective. Sometimes we need to stick to the plan, even if it doesn't survive first contact.

Testing & Quality Assurance

At the beginning of the project we were guilty of the cardinal sin of Engineering, not testing our work until the last minute. If the code compiled and the H Bridge didn't explode everything was dandy, testing was limited, avoiding stress testing the buggy and its code until the final acts of this module. Many small issues could have been flagged and solved earlier, saving us a lot of stress if we devised and conducted a full testing procedure at every stage like we adopted for the Gold Challenge.

Ethical Issues

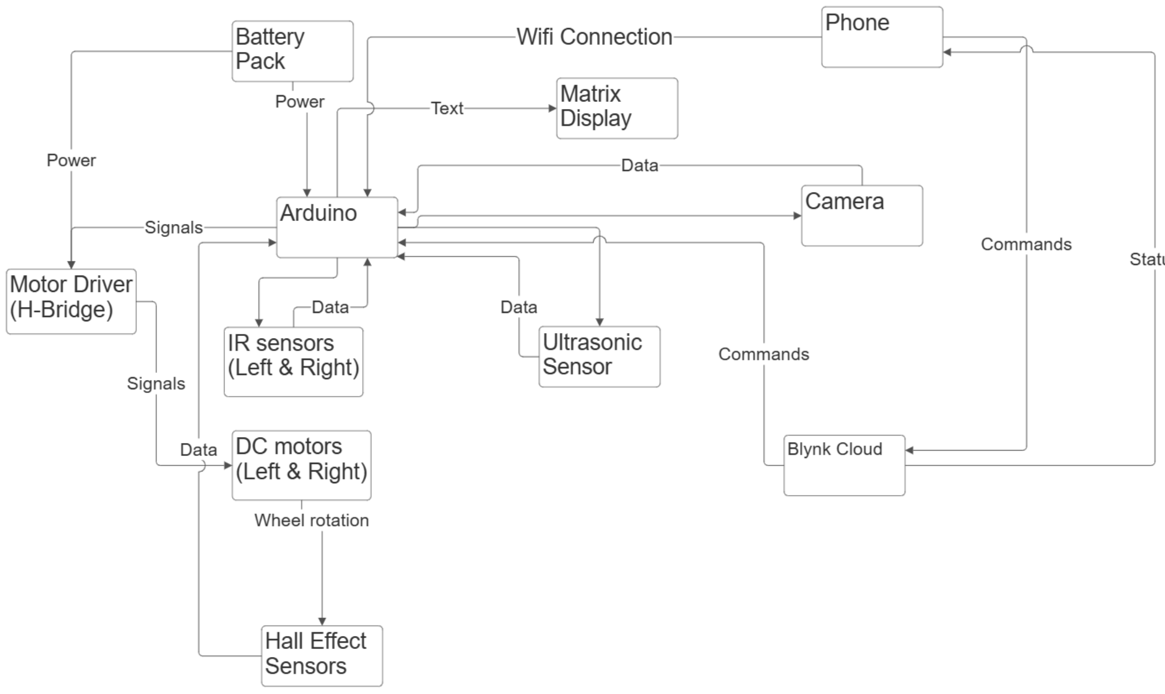
Overall, the ethical implications of this project are relatively mundane and are common to all semi-autonomous technologies. If one wanted to make a targeted weapon with this buggy it would be worryingly straight-forward. Simply adding a servo and taping a grenade to it along with its current capabilities would turn Maxine into a machine capable of serious damage. Especially with our added remote control system which was added during the gold challenge.

The data collected by the buggy carries little importance outside of its own operation. Our Buggy collects telemetry data and works on already known signals. We have not built a clever listening device.

Whilst using an internet connection to “talk” to the main computer running processing, there is no meaningful contact outside of this. The Buggy is a small computer on wheels, that is unless changes are made cannot and will not cause any damage to others or the environment in which it is found.

However we may have been rude to the Buggy, cursing and swearing at it when things didn’t go our way or the buggy behaved in a manner that we didn’t expect it to.

Part B: **Hardware Design**



Scenarios the buggy can handle:

Normal operation: To traverse around the track smoothly while staying on the white line during straights and turns.

Obstacle Detection and Following: If an object gets too close, the buggy must be able to stop immediately to avoid a collision. For the Silver challenge Maxine learned not only to halt in front of an obstacle but to match the speed of an object in front of it.

Manual override: If a stop or a go command is sent to the Arduino externally by a user, the buggy should immediately stop or go as requested.

Communication: The buggy is able to communicate distance travelled and when an object is detected to the GUI on processing.

Object recognition: The HUSKEYLENS camera can recognise objects within its lens and can also interpret road signs shown as tags. On our processing we stored the numbered tags as the Irish road signs.

Remote Control: Remote control via a GUI on a phone was added as a feature to the buggy via Blynk, an IOT platform. This enabled total manual control of the buggy and the ability to switch between the processing GUI and the phone's GUI seamlessly.

Functionalities and Override signals

Battery Pack: Supplies power to the Arduino and the motor driver (H-bridge).

Arduino: Acts as the central controller, processing input from sensors, responding to external commands and controlling motors.

Motor Driver (H-bridge): Receives signals from the Arduino to control the speed and direction of the two DC motors. The Arduino sends PWM signals to the H-Bridge, allowing forward, backward, left, and right movement by varying the voltage applied to the motors.

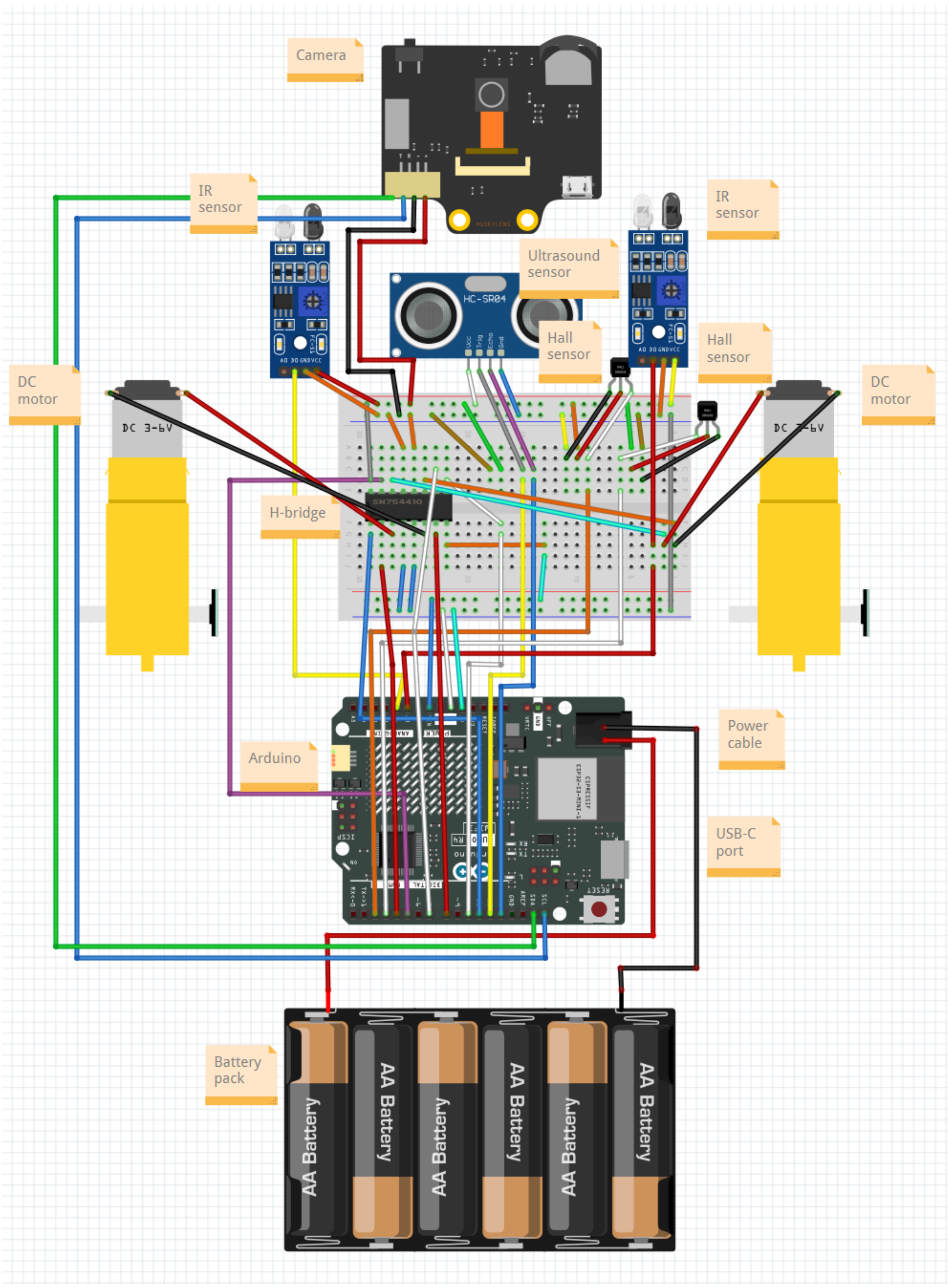
DC Motors: Drive the buggy forward, backward, or turn based on the H-bridge signals.

IR (infrared) Sensors: Used for line following, providing feedback to the Arduino. If a sensor detects the white track line, feedback is sent to the Arduino, adjusting the motor speeds resulting in a turn in a desired direction to stay on the track.

Ultrasonic Sensor: Measures distance to obstacles and sends data to the Arduino. If an obstacle is detected, an override signal is sent to the Arduino to stop the motors immediately, and the Arduino processes this input and stops the motors to avoid a collision.

Hall Effect Sensors: Monitor wheel rotation for speed or position feedback, feeding data into the Arduino. This allows us to calculate wheel speed and distance travelled.

Camera: Used for tag recognition and interpretation for gold challenge.


Camera
IR sensor
IR sensor
Ultrasound sensor
HC-SR04
Hall sensor
Hall sensor
DC motor
DC motor
DC 3-6V
DC 3-6V
H-bridge
SN754410
Arduino
Power cable
USB-C port
Battery pack
AA Battery
AA Battery
AA Battery
AA Battery
AA Battery
AA Battery

Overall Layout Utilisation

The breadboard was rewired after completing the silver challenge in an effort to make the buggy and breadboard connections as neat as possible . We tried to reduce the same colour wires overlapping where we could, to avoid confusion as to where the wire led. We also used small wires where possible, such as the connections to ground and the 5V power supply to avoid wires tangling or overlapping. small wires also had a practical use, they don't fall out easily which is helpful if one was to accidentally turn the buggy on if a certain element wasn't grounded.

The H-Bridge was put to the left hand side to clear up space on the right, this helped us greatly in being able to trace the connections between wires and also shortened some connections, allowing us to use smaller wires. The additional space made from these changes was filled with connections to the camera.

Challenges faced

The challenges we faced into turning our design into working hardware were not major challenges but rather minor speedbumps such as the accidental frying of an H-Bridge due to a grounded wire falling out and being put into a power pin accidentally, but we learned to use smaller wires from this, leading to a positive learning outcome. Another issue was the accidental bending of an H-Bridge pin which set us back timewise (as we initially believed it was a code issue) before taking out and inspecting the H-Bridge only to find one faulty pin. Other than these two minor incidents the wiring of the board and its elements did not represent a major challenge. Rewiring it for neatness did not raise any issues as it was simply replacing wires. Adding the camera to the board was also easy as it was only 4 connections, for which we had made space already.

Another challenge faced was the motors for the wheels. From the start, there was a power balance issue where supplying the motors with the same PWM value led to different RPMs on the individual motors. We initially fixed this via coding the weaker motor to have a higher PWM value; however, after Silver, the weaker motor became far too weak and needed to be replaced. The problem didn't completely disappear after this fix however, and we did face issues with ensuring the motors could cause the buggy to rotate 180 degrees correctly (see section D) as small changes in the battery charge level created significant power imbalances. Nonetheless, this was corrected by ensuring that the buggy would over-rotate the motors at all times via adding a longer delay to the code, a rudimentary but effective solution. This point certainly serves to be a source of improvement by implementing a more efficient spin feature that would use the IR sensors to ensure the buggy has fully rotated.

Part C: **Software Design**

The following outlines the codebase developed and the design process followed in the Silver Challenge. The software system integrates embedded PID control with a user-facing GUI to allow real-time interaction, tuning, and monitoring.

System Architecture

The software comprises two main components:

- **Arduino-Based Controller** – Responsible for executing control algorithms (distance and velocity PID), acquiring sensor data (ultrasonic and encoder), and driving actuators (motors).
- **Processing-Based GUI** – Provides an interface for issuing commands, adjusting parameters (e.g., velocity), and viewing telemetry data such as distance readings and encoder feedback.

System Features

- **Dual PID Controllers** for both distance and velocity regulation
- **Sensor Interfaces** including ultrasonic proximity sensing and quadrature encoder input
- **Actuator Control** via H-Bridge motor drivers using digital PWM
- **Command Communication Protocol** over Wi-Fi (TCP client-server)
- **Real-Time Feedback and Data Logging** to aid diagnostics and user interaction

Firmware Design on Arduino

PID Controller Implementation

Two PID controllers are implemented on the Arduino:

- **Distance PID**: Maintains a fixed distance from obstacles using ultrasonic sensor input.
- **Velocity PID**: Ensures wheel RPM matches user-defined target via the Processing GUI using encoder data.

These controllers are implemented with techniques for noise resistance and stability, such as anti-windup and adaptive timing.

Initialization Function (PIDsetup)

Sets up PID gain values and initializes state variables:

```
void PIDsetup(float Kp, float Ki, float Kd) {
    pid.Kp = Kp;
    pid.Ki = Ki;
    pid.Kd = Kd;
    pid.out_min = 0;
    pid.out_max = 255;
    pid.integral = 0;
    pid.prev_error = 0;
    pid.prev_measurement = 0;
    pid.prev_time = millis();
}
```

Control Computation (PIDcalculate)

Computes the output control signal based on the PID formulation:

```
float PID_Compute(PID* pid, float setpoint, float measurement) {
    unsigned long now = millis();
    float dt = (now - pid->prev_time) / 1000.0;
    if (dt <= 0) dt = 0.01;

    float error = setpoint - measurement;
    float P = pid->Kp * error;

    pid->integral += error * dt;
    pid->integral = constrain(pid->integral, pid->out_min, pid->out_max);
    float I = pid->Ki * pid->integral;

    float D = pid->Kd * (measurement - pid->prev_measurement) / dt;

    float output = P + I + D;
    output = constrain(output, pid->out_min, pid->out_max);

    pid->prev_error = error;
    pid->prev_measurement = measurement;
    pid->prev_time = now;

    return output;
}
```

Algorithmic Enhancements

- **Anti-Windup**: Integral term is clamped within bounds
- **Derivative-on-Measurement**: Reduces noise sensitivity
- **Time-Adaptive Control**: Responds consistently under variable loop timing

Main Control Loop

The main Arduino loop interprets incoming commands and executes the appropriate control strategy:

```
void loop() {
    if (command == 't') {
        float dist = readUltrasonic();
        float output = PID_Compute(&distPID, targetDist, dist);
        setMotorSpeeds(output);
    } else if (command == 'u') {
        int rpm = measureRPM();
        float output = PID_Compute(&velPID, targetRPM, rpm);
        setMotorSpeeds(output);
    }
    sendTelemetry();
}
```

Communication Protocol

A simple character-based TCP command system is used between the GUI and Arduino over a local Wi-Fi network. Commands such as "g" (go), "s" (stop), "t" (track distance), and "u" (track velocity) are transmitted from the GUI.

Telemetry data is sent back in plain text format:

```
ENC,1234
DIST,25.4
VEL,67
```

This allows real-time GUI updates with minimal parsing effort.

GUI Design (Processing)

Functional Overview

The GUI, built using Processing and the ControlP5 library, allows users to control the buggy wirelessly and monitor its behavior in real time.

Core Features

- **Live Telemetry**: Distance, velocity, and encoder data displayed
- **PID Activation Buttons**: Start/stop distance or velocity tracking
- **Speed Slider**: Dynamically sets the target RPM for the velocity controller
- **Command Logging**: Shows messages sent and received
- **Display of Institutional Branding** for submission identity

Data Handling

Incoming TCP messages are processed to update the GUI's displays:

```
void serialEvent(Client client) {
    String data = client.readStringUntil('\n');
    if (data != null) {
        String[] parts = split(data.trim(), ',');
        if (parts[0].equals("ENC")) {
            encoderValue = int(parts[1]);
        } else if (parts[0].equals("DIST")) {
            distance = float(parts[1]);
        }
    }
}
```

User Interface Components

- **Control Buttons**: "Go", "Stop", "Start Tracking", "Velocity Tracking"
- **Slider**: Adjusts velocity target dynamically
- **Log Area**: Outputs recent commands and telemetry messages
- **Distance/Odometer Panels**: Displays sensor readings and motion feedback

Conclusion

This software system integrates real-time embedded control with user-facing GUI tools for experimentation and demonstration. It enables precise movement through PID regulation and offers a clean, accessible interface for control and observation.

Key Outcomes:

- Reliable PID behavior for both obstacle tracking and speed regulation
- Smooth TCP communication with easy-to-parse message formats
- User-friendly interface for command dispatch and telemetry analysis

Future Enhancements:

- GUI-based PID gain tuning sliders
- Autonomous navigation routines with waypoint following
- Graphical plots of telemetry over time for performance evaluation

Part D: **Gold Challenge**

The following outlines the codebase developed and the development process for the Gold Challenge.

Challenge Goals

The gold challenge was broken down into the following goals to be accomplished via the software.

Main goal: Enable tag recognition to implement automated command processing for the buggy.

The commands which the buggy was to be able to achieve were:

- Go fast
- Go slow (a set speed achieved via PID)
- Turn left at a junction
- Turn right at a junction

As a further challenge, we chose to tackle the following goals as well,

- Stop for 5 seconds with a tag.
- Perform a Hairpin turn to the left and right (180 followed by a left or right turn at the next junction).
- Remote control via IOT with a phone

System Architecture

The developed system consists of three principal components:

1. **Arduino-Based Embedded Controller** – Responsible for executing the PID (Proportional, Integral, Derivative) algorithm, acquiring sensor data, managing actuator output, receiving remote control commands and executing them.
2. **Processing-Based GUI** – Facilitates real-time monitoring, control input, and data logging.
3. **Phone GUI** – Real-time remote control of buggy via forward, back, left and right buttons. Also, implement a method to switch back and forth from processing GUI control.

Base Gold Challenge Design

The following methodology was used to enable these features to achieve these objectives.

The Huskylens is an AI camera and is able to learn April tags. The April tags can be used to differentiate between the different commands which we wish the buggy to be able to perform. Due to the simplicity of the Huskylens user interface, the camera simply needed to be set to tag recognition mode and the button on top pressed while aiming at a tag until the UI confirmed that the tag had been saved. This was repeated, and the tag identification was saved locally on the lens.

The following code was used to recognise the tag on the Arduino:

```
void cameraloop() {

  if(millis() > current_time_c){

    if (huskylens.request()) {

        if (huskylens.count()) {

            for (int i = 0; i < huskylens.count(); i++) {

                HUSKYLENSResult result = huskylens.get(i);
                size_irl = result.width * result.height;

                Serial.println(size_irl);


                if(size_irl > 3000){ ID = result.ID; }

            }

        } else{ ID = 0; }

    }
```

The Huskylens cannot control the behaviour of the buggy by itself, instead, the I2C communication protocol was used to communicate the data from the Huskylens to the Arduino and vice versa. Note that the Arduino only communicates with the Husky lens to initialize the mode that it is in and to request the data the Huskylens reads.

- Processes Huskylens data

The Husky lens is capable of sending a tremendous amount of data to the Arduino; however, for our purposes, the only significant data was what tag it was reading, such that the Arduino can execute the command, and the amount of pixels the tag takes up on the camera so that the distance to the tag can be approximated.

- Execute the tag action

Now that the tag identifier is known, the Arduino must remember the tag ID and execute the task that the tag specifies at the correct location on the track.

- Communicate to Processing

As the Processing GUI is the method by which the user interacts with the Arduino, the tags read and the action being performed must then be communicated to the user. Hence, the Access point method from the ESP32 was used to establish a local network wherein the Arduino communicates its actions to the processing to alert the user before the action has even taken place.

We can explain this through pseudo-code as follows:

```
INITIALIZE Huskylens in tag recognition mode

Check if any tags have been learned.

Provide a connection error message in case Huskylens is not detected

LOOP:
```

```
REQUEST data from Huskylens
IF tag detected:
  READ tag ID and tag size
  IF tag size > threshold:
    STORE tag ID

MATCH tag ID to action:
  IF tag == fast:
    SET motor speed to max
  ELSE IF tag == slow:
    RUN PID controller to maintain target speed
  ELSE IF tag == left_turn:
    EXECUTE left junction logic
  ELSE IF tag == right_turn:
    EXECUTE right junction logic

UPDATE GUI via Processing:
  SEND current action and tag ID to Processing interface
```

Advanced Gold Challenge Design

This design enabled the implementation of the basic gold challenge objectives, the advanced features were implemented via the following methodology,

The first two were achieved by adding more tags, which the Huskylens learns and then can communicate with the Arduino to let the buggy perform the desired action. Remote control was implemented via Blynk, a low-level IOT platform that enables communication between a phone, the Blynk cloud server and the Arduino via the Blynk protocol. The phone communicates to the Blynk server via WiFi. The Blynk server then communicates to the Arduino. RC was implemented with the following steps,

- Set up a Thing (hardware-software interface) on the Blynk online platform

This step involves setting up a template where which pins on the Arduino are to be controlled by Blynk. Blynk has built in support for the Arduino devices, hence the support for the pins was already established. The Thing is connected to a Blynk account.

- Setting up the communication protocol

Blynk then generates an authentication token, template ID and device ID, which the Blynk library on the Arduino IDE can use to communicate with the Blynk server when connected to a WiFi network. Thus, the Blynk library was enabled, and the identifiers were added to the code. The Blynk library has two functions, Blynk.begin() and Blynk.run(). Blynk.begin() establishes the connection to the Blynk server from the Arduino, and Blynk.run() regularly checks for what commands are being sent from the phone.

- Creating the GUI on the phone

Blynk has a phone app which can be downloaded and signed into the Blynk account. The phone communicates to the Blynk server via WiFi. The account can then display the devices connected to the Blynk server, and a GUI can be created via buttons. The buttons communicate the behavior of the respective pins on the Arduino, which control the motor.

- Enable mode switching

To enable seamless switching between RC mode and processing, Blynk.begin() and Blynk.run() were set to be enabled when a specific command from processing was sent. The Arduino code then executes Blynk.begin() once to establish a connection and then loops Blynk.run() to enable the RC. Similarly, to switch back to control from processing, a switch command was also added to the Blynk thing, which commands the Arduino to exit the Blynk loop and restart the access point and enable the previous gold challenge abilities via processing.

As in the previous section we can explain the above through the following pseudo code:

```
SETUP Blynk with auth token and virtual pin layout

IF command_from_processing == "enable_RC":

  RUN Blynk.begin()

  WHILE in_RC_mode:

    Blynk.run()

    IF phone_button == forward:

      MOVE forward
    ELSE IF phone_button == left:
      TURN left
    ...
    IF phone_button == switch_to_processing:
      DISABLE Blynk
      RESTART Access Point mode
      RETURN to Processing control

ELSE:
  LOOP:
    REQUEST tag from Huskylens
    DETERMINE action from tag ID
    IF tag == stop:
      STOP motors
      WAIT 5 seconds
    ELSE IF tag == hairpin_left:
      EXECUTE 180° left turn, then follow with left junction
    ...
```

Part E: **Appendices**

A full record in addition to the folders in this submission found at this link here. This contains all the code for this project. All of which were listed here for the purposes of submission.

Appendix I: Processing Code

Can be found in submission in an arrangement of files listed as follows:

Gold_UI.zip and Silver_Challange_GUI.zip

Appendix II: Arduino Code

Two folders contain all relevant code for gold and silver challenge, RC.zip, Silver_AP_main.zip.

Appendix III: Development Log

Can be found in submission in file named “Appendix III”

Further to this please note that the Usernames of the group where as follows: Klion99: Guru Chappalle,

FLalor: Fintan Lalor,

& J3r1cho25: Joshua Searle