2E10 Final Gold Report

Group Y1

Developing a buggy robot relies on the creation of efficient control systems and algorithms. The main goal of the 2E10 project is to utilize the Arduino Uno R4 Wi-Fi microcontroller to create a versatile vehicle capable of navigating a track while being controlled via the Wi-Fi module which can detect different tags. This report presents an extensive overview of the project and how different sensors and cameras were implemented to develop the buggy and to complete a series of challenges.

Part A - Self-assessment

Three team strengths:

Resilience / Dedication
It is safe to say that our team had resilience. This project required lots of it. There were times where no matter what we tried, something seemed to go wrong. This came to climax when the morning of our scheduled silver demo, the buggy suddenly stopped working. The buggy that was working mere hours earlier could no longer even follow the track. Our team showed resilience and did not give up, relentlessly trouble shooting. As it turned out a surge on the Arduino board had caused one of the output pins to be constantly high. Also, out left wheel motor had locked up and capitulated. Despite all this we managed to gain an extension for the broken parts and refitted new motors and a new Arduino over the weekend, which also meant refitting the PID constants to the new motors. The next Monday, we successfully passed the silver challenge. Also, the day before the gold challenge, the buggy was too slow, and it took a while to react to different instructions. To fix this, we had to readjust our PID constants.
Organisation
Our team did immensely well in version control. Our shared GitHub ensured that everyone had the most up to date files and there was a clear path for development. Every challenge stage was given a new branch (bronzeDev, silverDev, goldDev), and reviewed as a team in a pull request at the end of development. This allowed us to clearly organise our goals for each development stage.

Communication
Our team was excellent at communication throughout the development process. We used WhatsApp to discuss ongoing concerns, such as anyone taking the buggy, when it was back in the shared locker, any upcoming issues that anyone needed help with, as well as scheduling in person meetings. In conjunction with this we used Slack which allowed us to assign specific issues to each member of the team. This was useful for accountability and keeping people on track.

Three team weaknesses:

Inflexibility
At times we suffered from too rigid of a team structure, where people did not venture outside their roles. We tended to stick to what each of us were doing in the first week for the most part. Not only did this prevent members from developing skills in all areas but also inhibited

collaboration and idea sharing down the line, as members did not understand other areas of the project as much as their own.

Failure Exhaustion
During the silver development stage, we suffered from failure fatigue. There was a string of issues that really tested our perseverance. We struggled to find suitable values for our PIDs. This was followed with immense difficulty in transporting numerical data across Wi-Fi from the Arduino to our Processing program because of the differences in integer/double encoding and string classes on Arduino. This was followed by the breakdown of one of our motors that forced us to re-tune our PID values. At this point it seemed that we wouldn't get to attempt gold, and everything seemed hopeless. Thankfully, we managed to pull through and we passed our gold challenge.

Lack of long-term vision
Initially during Bronze development, we struggled with short-term focus. Instead of building a good base that could be iterated upon for silver and gold, we aimed to solve the bronze challenge as quickly as possible. While this gave us some initial success, it meant that there was a lot of code that had to be rewritten/reevaluated during silver development that could have been avoided with a more long-term vision.

Three Changes to Gold Approach/Design that could improve quality of solution:

Our approach could have greatly benefited from designing around gold requirements from the very beginning. Although we were successful in completing every stage and getting to gold, there were instances where code written for bronze had to be completely redesigned for silver due to the implementation of the PID. This was not an issue progressing from silver to gold as we started to make more robust and long-thinking code. Had we done this from the start, we could have had more time to implement more creative gold features.

Our solution included April tags that were propped up by the roadside. This was an issue as it caused the buggy to detect and obstacle and go into follow mode. This meant that we had to disable follow mode when using April tags. If we could have maybe hung the April tags higher up than the US sensor or found some other way of avoiding the mode switching, this would have improved the final design.

We used the PID to slow down the buggy as it approached a speed limit tag, however out design could have been improved if we used size data from the husky lens to determine when we were at the sign.

Ethical issues

Data collection
The Husky lens which we have implemented for gold challenge has facial recognition, tag recognition and colour recognition features. Therefore, there is a risk of security issues and data collection issues related to this feature. As the buggy collects data about its surroundings and faces detected, there's an ethical risk of that data being intercepted or accessed by unauthorized parties if proper security measures are not implemented. There's a risk that the camera-equipped buggy could be repurposed for malicious activities, such as covert surveillance, harassment, or invasion of privacy. This could potentially lead to ethical issues such as misuse of sensitive information and privacy breaches.

Surveillance and Consent:

While the intention of using sensors like ultrasonic sensors may be for object detection and line tracking, there's a risk that the technology could be repurposed for surveillance purposes without consent, leading to potential infringement of privacy rights and other ethical risks.

The use of a camera-equipped buggy raises questions about surveillance and consent. People may not be aware that they are being recorded or may not consent to being captured by the camera. As a result, it's crucial to obtain consent when deploying such technology, especially in public places.

Ethical AI Use:
The use of computer vision algorithms for tasks such as object detection, face recognition, and tracking raise ethical concerns regarding algorithmic transparency, accountability, and potential biases. Developers must ensure that AI systems are deployed ethically, transparently, and with mechanisms for accountability and oversight.

Environmental Impact
Although it may seem less significant, even small devices like Arduino buggies can have an environmental impact, especially if they use non-renewable resources or contribute to electronic waste. Ethical considerations include minimizing environmental impact through responsible design, usage, and disposal practices.

As almost all of the components were manufactured abroad, there would have been a significant CO_2 emission in oversees transporting. This could be reduced by using locally manufactured products which would be more eco-friendly in terms of CO2 emissions, and it would support our local economy.

Bias and Discrimination: Computer vision systems, including those onboard the HuskyLens, are susceptible to biases and inaccuracies, which can lead to discriminatory outcomes. Developers must address biases in the training data and algorithms to ensure fair and equitable treatment of all individuals.

Part B - Hardware design

IR Sensors:

The two IR (Infrared) sensors play a crucial role in the functionality of our buggy. These sensors use an infrared emitter (IR LED) to transmit infrared waves on a surface. A photodiode receiver is then used to detect the reflected waves. By continuously monitoring sensor readings, we can implement a series of instruction loops to adjust the position of our buggy by detecting a difference in the reflected waves when it crosses the white track. This allows us to ensure that the buggy remains on the track and executes turns effectively.

The power (Vcc) and ground of the sensors are connected to the power and ground rail of the breadboard while the input pins are connected to the digital pin 3 and 4 on the Arduino Uno R4 Wi-Fi microcontroller.

This sensor uses a polling mechanism to continue the instruction (code) loop and make decisions based on the sensor reading. When the sensor reads a dark or black surface, it overrides the signals to ensure staying on track. The functions startMoving(),stopMoving() and checkTurn() enable the override signal in the circuit.

Ultrasonic Sensor

The Ultrasonic sensor is equipped with 4 different pins, such as, ground (GND), power (Vcc), echo and trigger (Trig). Upon activation, the trigger pin emits ultrasonic waves for a specific duration (10 ms) while the pin is set to HIGH. The echo pin will go HIGH when it detects the reflected waves. It then calculates the distance to the obstruction by using the time taken for the wave to return (pulseIn() function) and using the speed of sound (343 m/s). Whenever the echo pin detects an obstacle within 10 cm of the sensor, it uses interrupts to override signal to the Arduino to stop the buggy to avoid collision. The function checkObstruction() enables the override of the signal.
The power (Vcc) of the sensor is connected to the 5V pin on the Arduino and the ground is connected to the ground pin of the Arduino. Trig and echo are connected to digital pin 9 and 8 respectively.

To regulate and mitigate the error in the buggy's speed, a PID (Proportional Integral Derivative) controller was used for the silver challenge. The proportional term is the difference between the actual value and the desired value, while the integral error accumulates the total error over time and then sum is multiplied with a constant. For the Derivative, the rate of change is multiplied with the derivative gain to acquire the most accurate feedback.

$$u(t) = K_p e(t) + K_i \int_0^t e(t')dt' + K_d \frac{de(t)}{dt}$$

The constants were tuned using test and debug method. The PID controller was programmed such that the buggy's speed could be changed using the GUI by entering a reference speed and such that the buggy adjusts its speed to maintain a constant distance of 15cm from an object moving around the track in front of the buggy. The PID controller works by taking polling readings from the ultrasonic sensor and adjusting the buggy's speed (PWM) to reduce the total PID error which in return keeps the buggy's distance from a moving object constant.

DC Motors

The DC motor uses the fundamentals of electromagnetism to operate. As the armature rotates, it generates torque within the components, resulting in power generation. The maximum power supply of the DC motor used in the project is 6V.

The positive and negative end are connected to the input and output pins of the H-bridge to ensure speed and bi-directional control of the buggy. PWM is used to adjust the speed of the buggy by changing the pulse width to get varying analog values. PWM allows the buggy to achieve analog results with digital means by varying the on time of a square wave. The analogWrite() function is used to write a PWM value to the motor pins on the Arduino to vary the speed of the left motor and right motor.

Wheel Encoder:

Wheel encoder calculates the number of times the motor has rotated which can be used to calculate the distance travelled by the buggy. The wheel encoder consists of a rotating ring magnet which changes the magnetic fields as the motor turns, and a hall sensor to measure the strength of this magnetic field. In the wheel encoder, there is a photodetector which counts each rotation of the wheel, and this is converted into a signal. The signal is transformed into electric pulse, and we can measure the distance. Each wheel encoder contains 8 magnets to change the magnetic field around it. After we calculate the average revolution of the wheels, the total distance can be calculated using the wheel circumference.

Avg_revolution= $\frac{0.5\times(right\_wheel\_distance+left\_wheel\_distance)}{8}$

Distance Travelled= Avg_revolution X wheel Circumference

H-Bridge

A H-bridge serves as a power amplifier, facilitating the control of the buggy's speed and enabling bi-directional movement. It also ensures efficient power supply to the motors. The H-bridge is comprised of four switches and uses PWM (Pulse Width Modulation) and digital logic circuits to control the motors. The first pins on both ends are called the enablers, which are connected to the Arduino digital pins 5 and 6. The next pins are the input pins which are connected to the Arduino digital pins 11 and 12. Then the positive terminals of the motors are connected to the H-bridge on both sides. The ground pins are the 4th and 5th pins of the H-bridge. After that, the 6th pins are connected to the negative terminals of the motor. Finally, the last pin on one side is connected to the Cable Mount DC Jack Socket, ensuring efficient power supply by bypassing the power directly to the H-bridge.

The HuskyLens Camera

The Huskylens camera was integrated into the buggy's design as part of the gold challenge to identify markers on the track and make decisions based on what it detects. Aswell as being a camera, the Huskylens also contains an onboard computer vision system which allows implement object detection for the gold challenge. The Huskylens' onboard hardware also supports tasks such as object tracking and recognition, face recognition, line tracking, colour recognition, and tag recognition.

The Huskylens is connected to the Arduino very simply by the I2C protocol. The Green (TX) pin of the HuskyLens is connected to the SDA and the Red (RX) line to SCL, to use I2C.The HuskyLens is also connected to the 5V power line and the ground rails...

For this project, the Huskeylens was trained to recognise April tags to speed up or slow down, and to turn left or right. Four different tags were used to implement this. The camera was trained to recognise the tags and using the detectTags() function. The reference speed was changed to speed up or slow down the buggy, while a Boolean case was used to turn right or left.

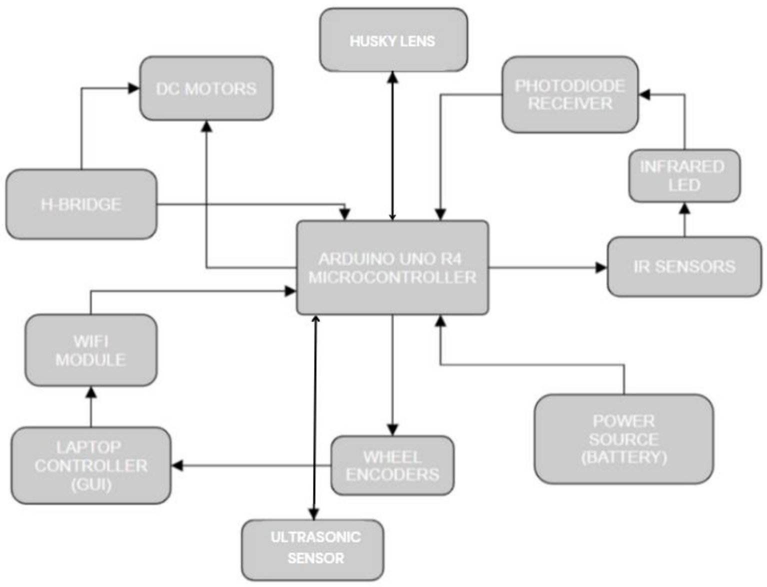

Fig.1: Block Diagram of the Buggy Components

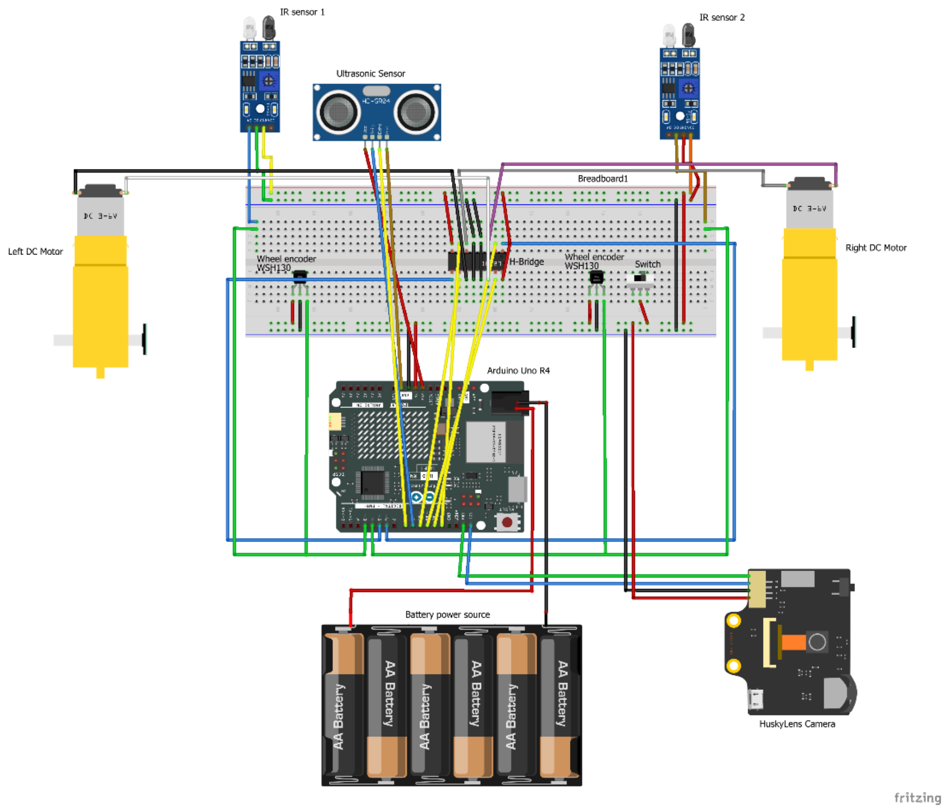

Fig.2: Breadboard Buggy Layout from Fritzing Software (above)

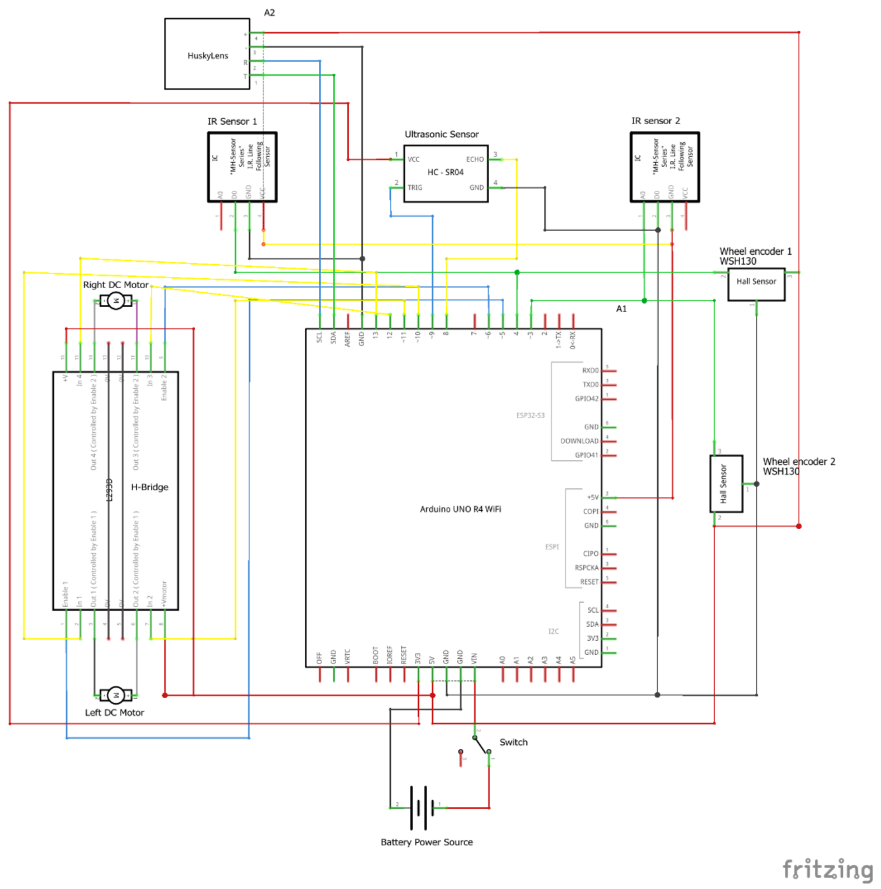

Fig.3: Schematic of the Buggy from Fritzing Software

Part C (Silver)

Buggy Hardware Interface:

The buggy hardware interface can be thought of in four main functional blocks:

1. Turning detection and controls.
2. Obstruction detection and controls.
3. Speed control.
4. Event reporting.

1. Turning detection and controls.

The infra-red sensors at the front of buggy detect changes in the colour (differences of radiation absorption) of the floor as the buggy moves. In our buggy, the program notes the starting state of the IR sensors. The sensors send a digital 'high' (1) when it sees a bright background, or a digital 'low' (0) when it sees a dark background. Every loop iteration, the value of these sensors is read to two of the digital pins on the Arduino. If the state of the sensor is different to the initial state, a turning action is required. As such, the Arduino sends analogue instructions to the wheel motors via the H-bridge from two of the pulse-width modulation pins on the Arduino. If a state change was detected on the right IR sensor, the left motor would receive a higher analogue value of 190/255 (higher speed), and the right motor would receive a lower analogue value of 15/255 (lower speed), causing the buggy to turn to the right.

When that is completed, the loop begins again, if there is no longer a difference in initial sensor reading and current sensor readings, the Arduino sends even analogue signals to the motors via the PWM pins and H-bridge i.e. 150/255 to both, causing the buggy to drive straight again.

2. Obstruction detection and controls.

The ultra-sonic at the front of the buggy is activated every 15 iterations to detect obstacles. The ultra-sonic sensor sends out a pulse of sound and waits for it to hit something and come back. When it does, the distance of the object is calculated. If the object is less than 10cm away, the Arduino updates the states of the digital control pins, to turn off both motors, via the H-bridge.

The US sensor also calculates the distance to an object that the buggy should follow. This value is supplied to a PID which updates the speed of the buggy.

3. Speed control.

In follow mode, the buggy tries to keep the distance to the object in front constant. The PID function is passed the distance to the object as well as the set-point (follow distance). If the buggy is too far behind or too close, the PID will update the PWM setting to speed up or slow down the buggy.

In reference mode, the buggy follows a reference speed set by a dial in the GUI. This reference speed is sent over Wi-Fi to the Arduino. The PID function is sent the reference speed as the set-point, as well as the current speed of the buggy. The PID adjusts the PWM setting to match the reference speed.

4. Event reporting.

To receive instructions and report events, the Arduino creates a wireless access point and a server, which is connected to by the supervisor PC. Instructions are sent on the form of short strings, for example “w” for go or “s” for stop. These messages are written to the server (Arduino) when a click of the stop or go button is detected in processing. The Arduino also sends information back to the supervisor PC again in the form of short strings such as “O” for obstruction detected, this string is written to the client (PC) when the obstruction loop detects a nearby obstruction. Telemetry data such as buggy speed (coming from the wheel encoders), distance travelled, etc. is also sent.

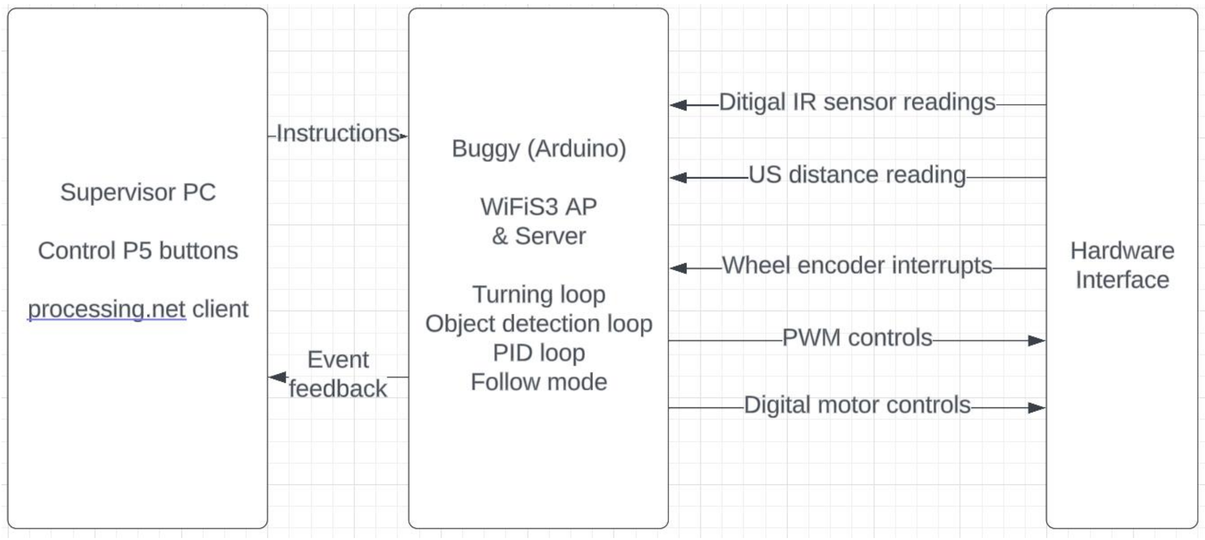

Fig 4. Silver buggy system model.

Control Flow Algorithms:

Supervisor PC pseudo-code in Processing:

```
import controlP5.*;
import processing.net.*;
//import meter.*;
ControlP5 p5;
Button GOButton;
Button STOPButton;
Textfield refSpeedTextbox;
Knob refSpeedKnob;

Client myclient;
//strings to be displayed
String data, obs_seen = "clear",
control_string = "Control mode: distance PID.",
distance_string = "Distance travelled: ",
bs_string = "Buggy speed: ",
us_string = "Distance to buggy: 0cm";

boolean control_mode = true;
PFont font, pFont;
int distance =   0;

void setup() {
    //init
    size(1600, 1000);
    background(0);
    pFont = createFont("Arial",16,true);
    font = createFont("calibri", 25);

    p5 = new ControlP5(this);

    GOButton = p5.addButton("buggy ON");
    STOPButton = p5.addButton("buggy OFF");
    refSpeedKnob = p5.addKnob("Speed");

    myclient = new Client(this, "192.168.4.1", 5200);
}

void draw() {
    background(0);
    //sont settings
    textFont(pFont,40);
    fill(255);
    //display metrics
    text(bs_string,900,250);
    text("Obstruction: "+obs_seen,900,550);
    text(control_string,900,850);
    text(distance_string,900,675);
    text(us_string,900,375);
    data = myclient.readString();
    try{
        if (data.equals("+")){
        obs_seen = "Clear.";
        }

        if(data.equals("O")){
        obs_seen = "Obstruction!";
        }

        if(data.startsWith("d")){
        distance_string = "Distance travelled: " + data.substring(1) + "cm";
        }

        if(data.startsWith("bs")){
        bs_string = "Buggy speed: " + data.substring(2) + "cm/s";
        }

        if(data.startsWith("us")){
        us_string = "Distance to buggy: " + data.substring(2) + "cm";
        }

        if(data.equals("01")){
            control_string = "Control mode: reference speed.";
        }else if(data.equals("00")){
            control_string = "Control mode: distance PID.";
        }
    }catch(Exception e){
    }
}

public void controlEvent(ControlEvent event) {
  if (event.isController()) {
    if (controller_name.equals("buggy ON")) {
      println("Buggy ON button pressed");
      myclient.write("w");
    }
    if (controller_name.equals("buggy OFF")) {
      println("Buggy OFF button pressed");
      myclient.write("s");
    }
    if(controller_name.equals("Speed")){
      float sliderValue = controller.getValue();
      myclient.write('n');
      myclient.write(sliderValue);
    }
    }
  }
```

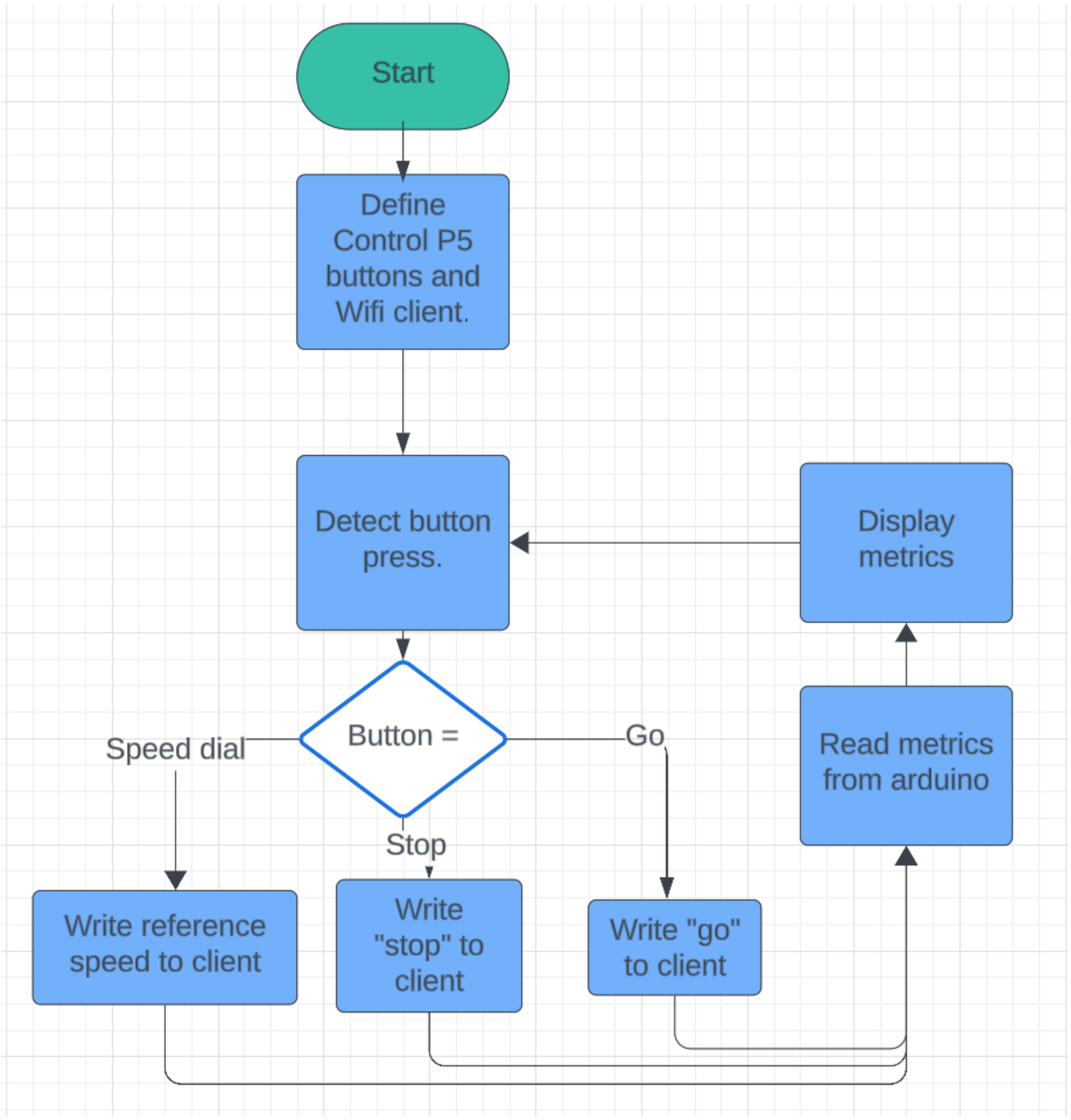

Fig 5. Processing Silver Control Flow

Arduino Silver pseudo-code:

```
#include <WiFiS3.h>
//network settings
char ssid[] = "Y1_AP";
char pass[] = "bababooey";
WiFiServer server(5200);
WiFiClient client;

//setting up sensor pins, global vars, constants
const int LEYE = 7;
//…


void setup() {
  beginSerial();
  beginWiFi();
  defineWheelInterrupts(functions = updateRwEnc, updateLwEnc);

  track_colour = digitalRead(LEYE);
  startMoving();
}

void loop() {
  cycles++;
  client = server.available();

  if(client.available()){//if client has message to read
    instruction = client.read();

    if(instruction == 'w' || instruction == 's'){
      carry_out = instruction;
    }

    if(instruction == 'n'){
      String sinstruction = client.readStringUntil('n');
      reference_speed = client.read().toFloat();
    }
  }

  if(carry_out == 'w'){//stop or go instruction from GUI
    keepMoving = true;
  } else if(carry_out == 's'){
    keepMoving = false;
    stopMoving();
  }

  if(keepMoving == true){

    if(control_mode == follow){//distance follow control mode
      computePID(distance);
      checkTurn();
      checkObstruction();
      distanceReport();

    }else if(control_mode == reference){//reference speed control mode
      speedComputePID(buggy_speed); //speed based PID
      checkTurn();
      checkObstruction();
      distanceReport();
    }
  }else {
    stopMoving();
  }

  //calculate speed
  if(2_seconds_passed){
    buggy_speed = (distance_travelled - previous_distance_travelled)/2;
    client.write(buggy_speed);
  }
}

void startMoving(){
  setMotorDirectionAndSpeed();
}

void stopMoving(){
  stopMotors();
}

void checkTurn(){

  readIRSensors();

  //track_colour is bckgnd and not line colour
  if(leftEye != track_colour && rightEye == track_colour){
      turnLeft();
  }else if(rightEye != track_colour && leftEye == track_colour){
      turnRight();
  }else{
      continueStraight();
  }
}

void checkObstruction(){

    if(cycles%15 == 0){
    distance = calculateDistanceToObject();
    if(distance<10){
      stopMoving();
      if(!same_obstruction){
        client.write("O"); //inform client of obstruction
        same_obstruction = true;
      }

    }else{
      startMoving();
      same_obstruction = false;
      client.write("+");//inform moving off
    }
  }

  if(cycles%50==0){ // every 50 cycles send distance to object ahead
    client.write(distance);
    if(distance > 70){//if no objects in range switch to reference speed
        control_mode = 1;
        client.write("01");//inform client of mode change
    }else if(distance < 70){//otherwise switch to follow mode
        control_mode = 0;
        client.write("00");
    }
  }
}

void updateRwEnc(){//add count to right wheel encoder
  r_wheel_d += 1;
}

void updateLwEnc(){//add count to left wheel encoder
  l_wheel_d += 1;
}


void distanceReport(){ //calculates distance travelled and informs client

  double avg_revs = (0.5 * (r_wheel_d + l_wheel_d)) / 8;//8 magnets ineach
wheel encoder
  distance_travelled = avg_revs * 20.6;//wheel circumference

  if(distance_travelled - prev_dist >= 36){
    prev_dist = distance_travelled;

    client.write(distance_travelled);

  }
}
```

```
void computePID(int US_read){//distance follow PID

  if(cycles % 15 ==0){

    error = US_read - set_point;//difference in follow distance & current dist
    cum_error += error * elapsedTime;                // compute integral
    rate_error = (error - last_error)/elapsedTime;   // compute derivative

    double out = kp*error + ki*cum_error + kd*rate_error; //PID o
    PWM_speed += out;

    if(PWM_speed < 70){//speed bounds for stability
      PWM_speed = 70;
    }else if(PWM_speed > 130){
      PWM_speed = 130;
    }

    global_speed = PWM_speed;//speed that gets sent to motors
  }
}


void speedComputePID(float SPEED_read){//ref speed PID

  if(2_seconds_passed){
     //dif between reference speed and actual
    error = SPEED_read - reference_speed;
    cum_error += error * elapsedTime;                // compute integral
    rate_error = (error - last_error)/elapsedTime;   // compute derivative

    double out = Skp*error + Ski*cum_error + Skd*rate_error;
    PWM_speed -= out;

    if(PWM_speed < 70){//speed bounds for stability
      PWM_speed = 70;
    }else if(PWM_speed > 150){
      PWM_speed = 150;
    }
    global_speed = PWM_speed;
  }
}
```

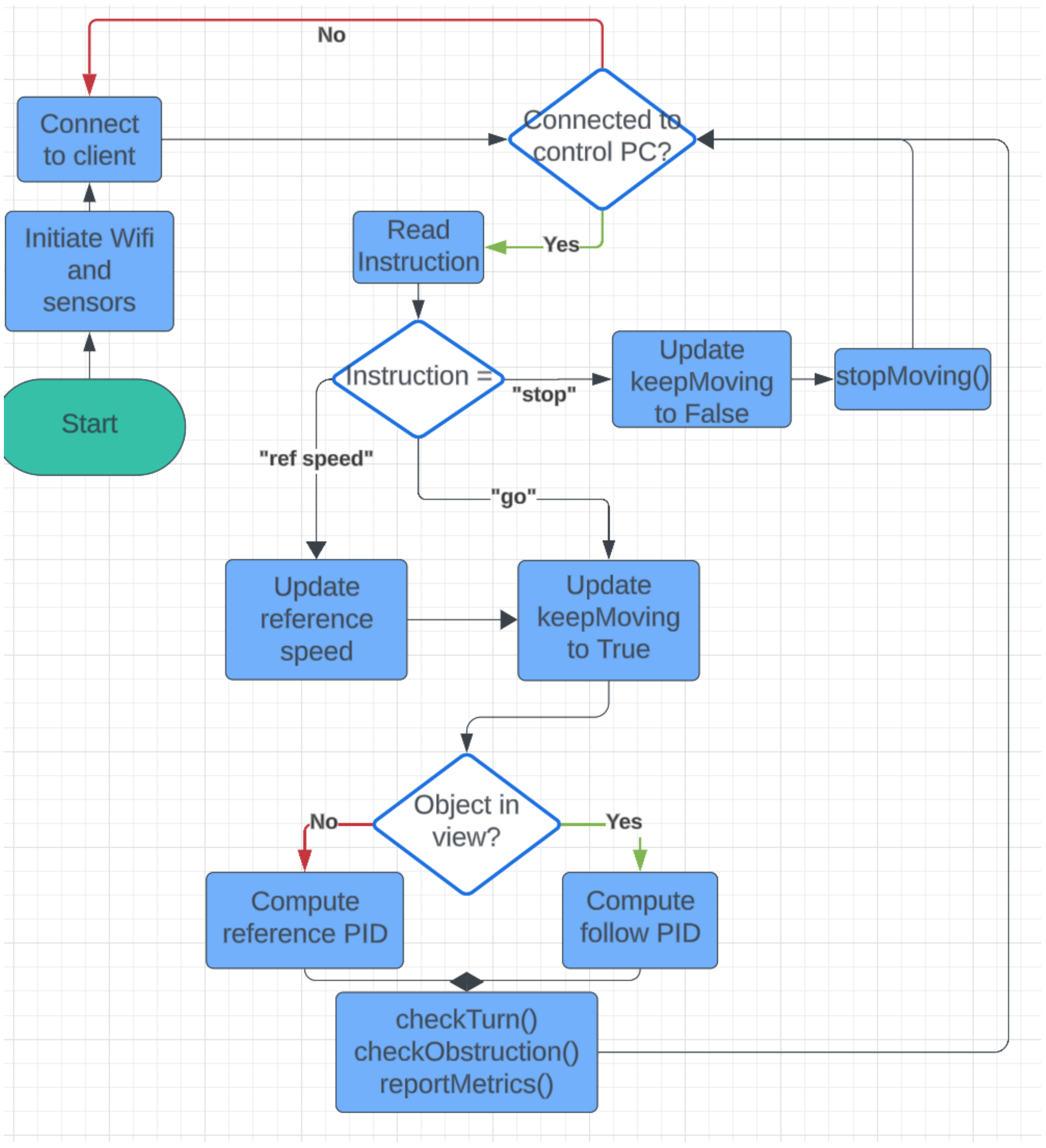

Fig 6. Arduino Silver Control Flow

Part D - Gold

Buggy Hardware Interface:
The buggy hardware interface can be thought of in four main functional blocks:

1. Turning detection and controls.
2. Obstruction detection and controls.
3. Speed control.
4. Event reporting.

1. Turning detection and controls.

The infra-red sensors at the front of buggy detect changes in the colour (differences of radiation absorption) of the floor as the buggy moves. In our buggy, the program notes the starting state of the IR sensors. The sensors send a digital 'high' (1) when it sees a bright background, or a digital 'low' (0) when it sees a dark background. Every loop iteration, the value of these sensors is read to two of the digital pins on the Arduino. If the state of the sensor is different to the initially recorded state, a turning action is required. The HuskyLens camera has been trained on 4 April tags, denoting a speed increase, speed decrease, left turn at next intersection, or right turn at next intersection. When a speed change tag is detected, the reference speed is updated, and wheel speed (PWM) is changed by the reference speed PID. In the case of a turn, the Arduino will wait until both IR sensors report a change in brightness and turn in the specified direction.

To turn, the Arduino sends analogue instructions to the wheel motors via the H-bridge from two of the pulse-width modulation pins on the Arduino. If a state change was detected on the right IR sensor, the left motor would receive a higher frequency value of 190/255 for example (higher speed), and the right motor would receive a lower frequency value of 15/255 (lower speed), causing the buggy to turn to the right.

When that is completed, the loop begins again, if there is no longer a difference in initial sensor reading and current sensor readings, the Arduino sends even frequencies to the motors via the PWM pins and H-bridge i.e. 150/255 to both, causing the buggy to drive straight again.

2. Obstruction detection and controls.

The ultra-sonic at the front of the buggy is activated every 15 iterations to detect obstacles. The ultra-sonic sensor sends out a pulse of sound and waits for it to hit something and come back. When it does, the distance of the object is calculated. If the object is less than 10cm away, the Arduino updates the states of the digital control pins, to turn off both motors, via the H-bridge.

The US sensor also calculates the distance to an object that the buggy should follow. This value is supplied to the follow PID which updates the speed of the buggy.

3. Speed control.

In follow mode, the buggy tries to keep the distance to the object in front constant. The follow PID function is passed the distance to the object as well as the set-point (follow distance). If the buggy is too far behind or too close, the PID will update the PWM frequency to speed up or slow down the buggy to maintain the follow distance.

In reference mode, the buggy follows a reference speed set by a dial in the GUI. This reference speed is sent over Wi-Fi to the Arduino. The reference speed PID function is sent the reference speed as the set-point, as well as the current speed of the buggy from the hall sensors (wheel encoders). The PID adjusts the PWM frequency to match the reference speed.

4. Event reporting.

To receive instructions and report events, the Arduino creates a wireless access point and a server, which is connected to by the supervisor PC. Instructions are sent on the form of short strings, for example “w” for go or “s” for stop. These messages are written to the server (Arduino) when a click of the stop or go button is detected in processing. The Arduino also sends information back to the supervisor PC again in the form of short strings such as “O” for obstruction detected, this string is written to the client (PC) when the obstruction loop detects a nearby obstruction. Telemetry data such as buggy speed (coming from the wheel encoders), distance travelled, etc. is also sent.

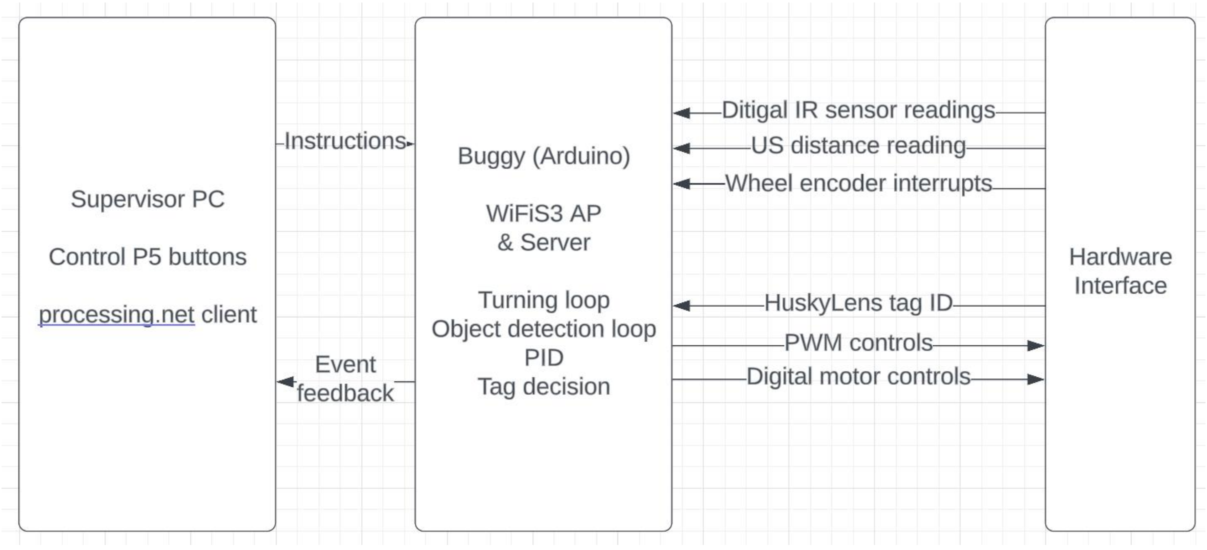

Fig 7. Gold buggy system model.

Control Flow Algorithms:

Supervisor PC pseudo-code in Processing:

```
import controlP5.*;
import processing.net.*;
//import meter.*;
ControlP5 p5;
Button GOButton;
Button STOPButton;
Textfield refSpeedTextbox;
Knob refSpeedKnob;

Client myclient;
//strings to be displayed
String data, obs_seen = "clear",
control_string = "Control mode: distance PID.",
distance_string = "Distance travelled: ",
bs_string = "Buggy speed: ",
us_string = "Distance to buggy: 0cm";

boolean control_mode = true;
PFont font, pFont;
int distance =   0;

void setup() {
    //init
    size(1600, 1000);
    background(0);
    pFont = createFont("Arial",16,true);
    font = createFont("calibri", 25);

    p5 = new ControlP5(this);

    GOButton = p5.addButton("buggy ON");
    STOPButton = p5.addButton("buggy OFF");
    refSpeedKnob = p5.addKnob("Speed");

    myclient = new Client(this, "192.168.4.1", 5200);
}

void draw() {
    background(0);
    //sont settings
    textFont(pFont,40);
    fill(255);
    //display metrics
    text(bs_string,900,250);
    text("Obstruction: "+obs_seen,900,550);
    text(control_string,900,850);
    text(distance_string,900,675);
    text(us_string,900,375);
```

```
  data = myclient.readString();
  try{
    if (data.equals("+")){
    obs_seen = "Clear.";
    }

    if(data.equals("O")){
    obs_seen = "Obstruction!";
    }

    if(data.startsWith("d")){
    distance_string = "Distance travelled: " + data.substring(1) + "cm";
    }

    if(data.startsWith("bs")){
    bs_string = "Buggy speed: " + data.substring(2) + "cm/s";
    }

    if(data.startsWith("us")){
    us_string = "Distance to buggy: " + data.substring(2) + "cm";
    }

    if(data.equals("01")){
      control_string = "Control mode: reference speed.";
    }else if(data.equals("00")){
      control_string = "Control mode: distance PID.";
    }
  }catch(Exception e){
  }
}

public void controlEvent(ControlEvent event) {
 if (event.isController()) {
  if (controller_name.equals("buggy ON")) {
   println("Buggy ON button pressed");
   myclient.write("w");
  }
  if (controller_name.equals("buggy OFF")) {
   println("Buggy OFF button pressed");
   myclient.write("s");
  }
  if(controller_name.equals("Speed")){
   float sliderValue = controller.getValue();
   myclient.write('n');
   myclient.write(sliderValue);
  }
  }
 }
```

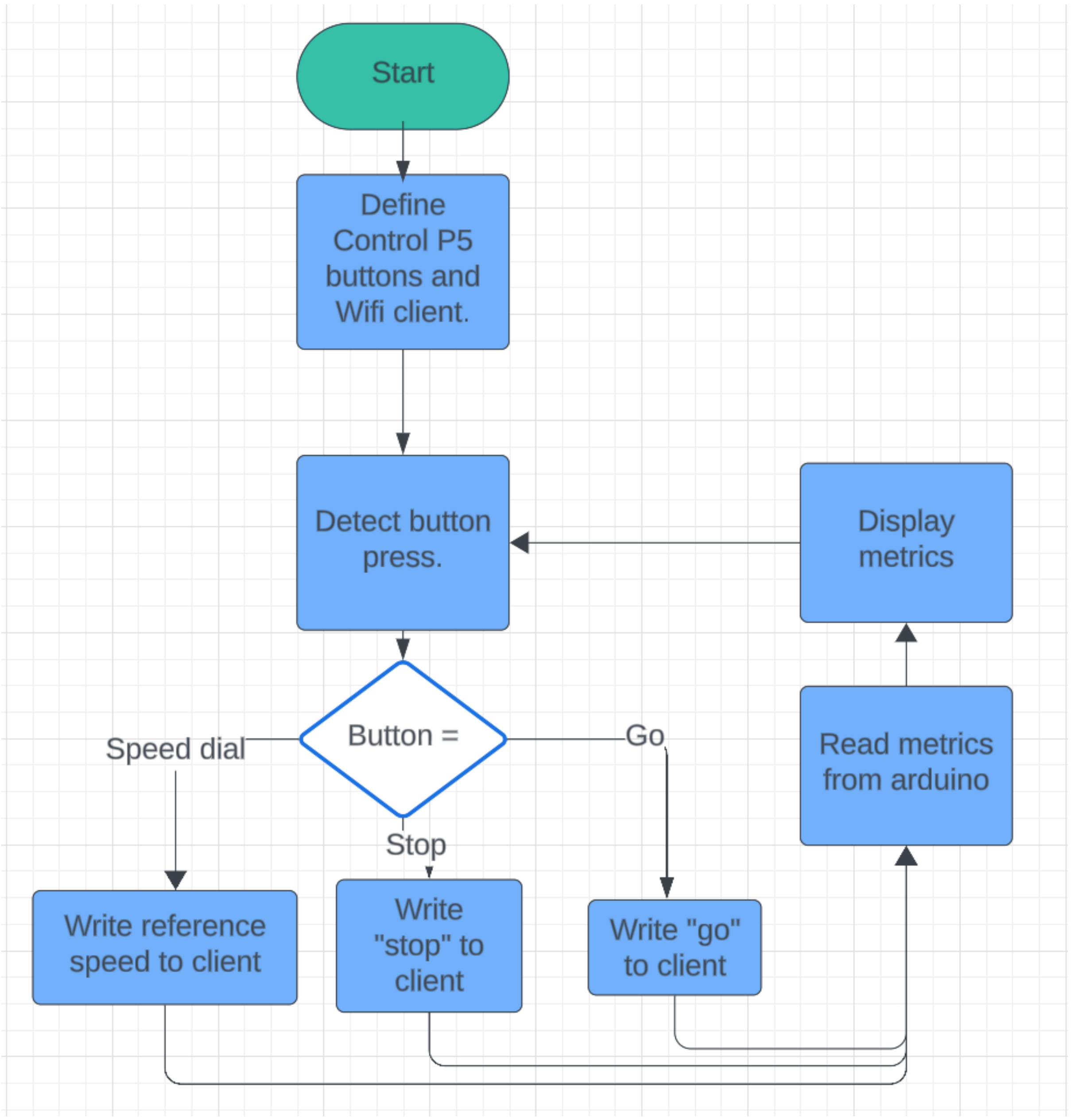

Fig 8. Processing Gold Control Flow

Arduino pseudo-code:

```
//computer vision camera lib
#include "HUSKYLENS.h"
#include "SoftwareSerial.h"

HUSKYLENS huskylens;
int tag_id = 0; //april tag id
bool turning_left = false, turning_right = false;

#include <WiFiS3.h>
//network settings
char ssid[] = "Y1_AP";
char pass[] = "bababooey";
WiFiServer server(5200);
WiFiClient client;

//IR sensors
const int LEYE = 7;
...


void setup() {
  beginSerial();
  beginWiFi();
  defineWheelInterrupts(functions = updateRwEnc, updateLwEnc);
  initialiseHusky();//start husky lens
  track_colour = digitalRead(LEYE);
  startMoving();
}

void loop() {
  cycles++;
  client = server.available();

  if(client.available()){//if client has message to read
    instruction = client.read();

    if(instruction == 'w' || instruction == 's'){//movement instructions
      carry_out = instruction;
    }
    if(instruction == 'n'){//reference speed set
      reference_speed = client.read().toFloat();
    }
  }

  if(carry_out == 'w'){//stop or go instruction from GUI
    keepMoving = true;
  } else if(carry_out == 's'){
    keepMoving = false;
    stopMoving();
  }

  if(keepMoving == true){
    speedComputePID(buggy_speed);
    checkTurn();
    checkObstruction();
    detectTags();
    distanceReport();

  }else {
    stopMoving();
  }

  //calculate speed
  if(2_seconds_passed){
    buggy_speed = (distance_travelled - previous_distance_travelled)/2;
    client.write(buggy_speed);
  }

}

void startMoving(){
  setMotorDirectionAndSpeed();
}

void stopMoving(){
  stopMotors();
}

void checkTurn(){

  leye_state = digitalRead(LEYE);
  reye_state = digitalRead(REYE);

  if(leye_state  != track_colour && reye_state != track_colour && turning_left
== true){
    turnLeft(); //intersection left turn
  } else if(leye_state  != track_colour && reye_state != track_colour &&
turning_right == true){
    turnRight(); //intersection right turn
  }else if(leye_state  != track_colour && reye_state == track_colour){//left
turn condition
    turnLeft();
  }else if(reye_state  != track_colour && leye_state == track_colour){ //
right turn condition
    turnRight();
  }else{//continue straight
    continueStraight();
  }
}

void checkObstruction(){

    if(cycles%15 == 0){
    distance = calculateDistanceToObject();

    if(distance<10){
      stopMoving();
      if(!same_obstruction){
        client.write("O"); //inform client of obstruction
        same_obstruction = true;
      }

    }else{
      startMoving();
      same_obstruction = false;
      client.write("+");//inform moving off
    }
  }

  if(cycles%50==0){ // every 50 cycles send distance to object ahead
    client.write(distance);
    if(distance > 70){//if no objects in range switch to reference speed
        control_mode = 1;
        client.write("01");//inform client of mode change
    }else if(distance < 70){//otherwise switch to follow mode
        control_mode = 0;
        client.write("00");
    }
  }
}

void updateRwEnc(){//add count to right wheel encoder
  r_wheel_d += 1;
}

void updateLwEnc(){//add count to left wheel encoder
  l_wheel_d += 1;
}

void distanceReport(){ //calculates distance travelled and informs client

  double avg_revs = (0.5 * (r_wheel_d + l_wheel_d)) / 8;//8 magnets ineach
wheel encoder
  distance_travelled = avg_revs * 20.6;//wheel circumference
  if(distance_travelled - prev_dist >= 36){
    prev_dist = distance_travelled;

    client.write(distance_travelled);

  }
}

void speedComputePID(float SPEED_read){//ref speed PID
  if(2_seconds_passed){
    //dif between reference speed and actual
    error = SPEED_read - reference_speed;
    cum_error += error * elapsedTime;                // compute integral
    rate_error = (error - last_error)/elapsedTime;   // compute derivative

    double out = Skp*error + Ski*cum_error + Skd*rate_error;
    PWM_speed -= out;

    if(PWM_speed < 70){//speed bounds for stability
      PWM_speed = 70;
    }else if(PWM_speed > 150){
      PWM_speed = 150;
    }
    global_speed = PWM_speed;
  }
}

void initialiseHusky(){
  Wire.begin();
  while( !huskylens.begin(Wire) ){
    delay(1000); // Wait a second before trying to initialise again.
  }
}

void detectTags(){//detects april tag id and updates instructions

  if (huskylens.request() && huskylens.isLearned() && huskylens.available())
  {
    HUSKYLENSResult result = huskylens.read();
    tag_id = (result.ID);
  }

  switch(tag_id){

    case 1://speed up
    reference_speed = 20;
    break;
    case 2://slow down
    reference_speed = 10;
    break;

    case 3: // turn left at next intersection
    turning_right = false;
    turning_left = true;
    break;

    case 4: // turn right at next intersection
    turning_right = true;
    turning_left = false;
    break;

    default:
    //Serial.println("No tag detected");
    break;
  }
  tag_id = 0;
}
```

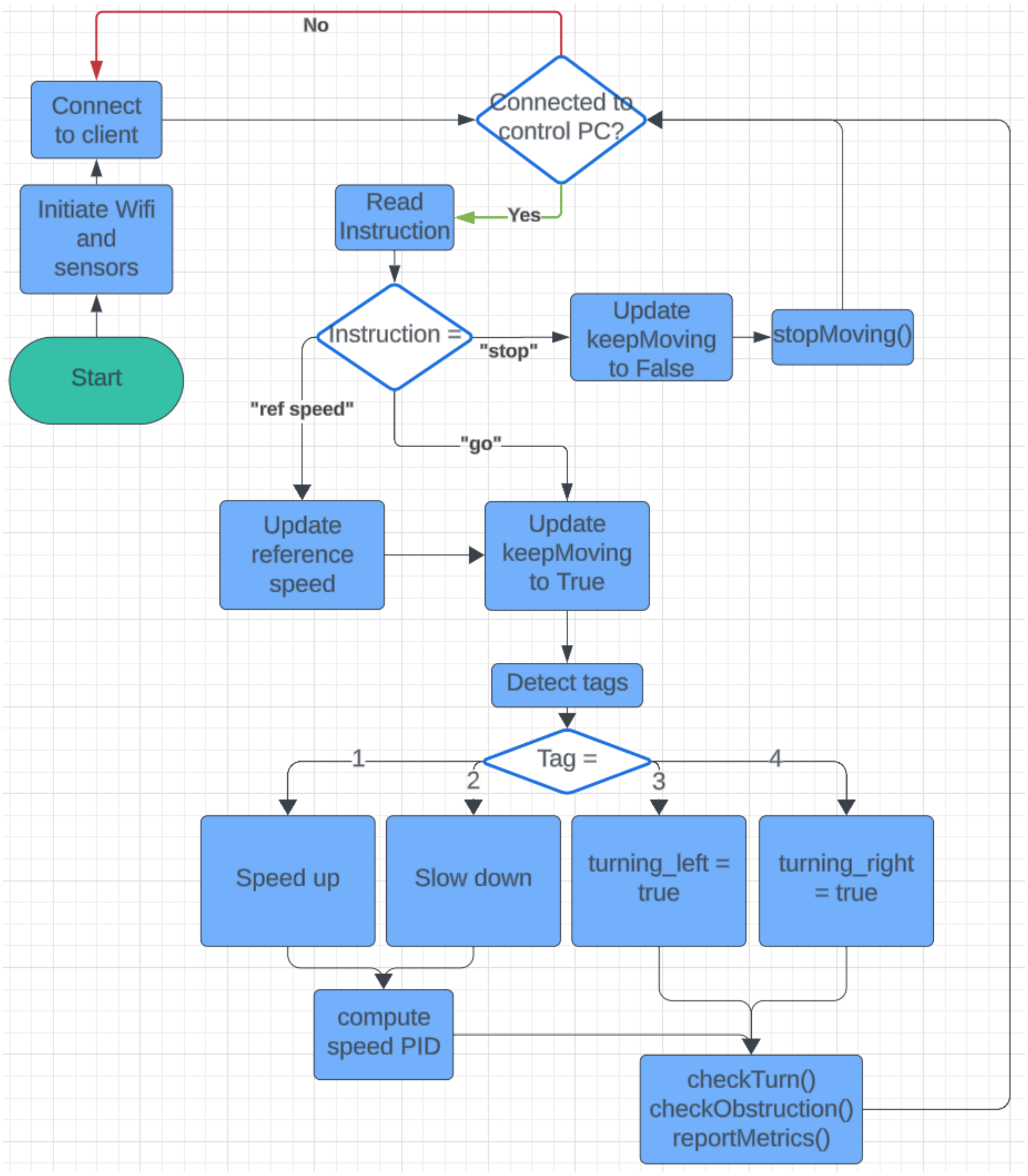

Fig 9. Arduino Gold Control Flow

CONCLUSION

In conclusion, the report reflects the development of the project using the sensors, camera and micro controller by showcasing the key elements of a buggy robot. The team have successfully completed all the challenges and this project was a great learning curve for us as it built a strong foundation of electronics and programming by utilising our problem-solving skills. Although there were many difficulties along the way, the team were able to learn from these and improvise to progress to the next stage. It also allowed us to learn and changed our way of approaching problems to consider the bigger picture and develop a solution with strong foundations that wouldn't need to be changed in the future. As we conclude this project, we are well equipped to tackle future projects with innovation and problem-solving.

Part E - Appendices.

Appendix I - PC processing code: gui_silver_vfinal.pde

Appendix II.I - Arduino Silver Sketch code: SilverBuggyVFinal.ino

Appendix II.II - Arduino Gold Sketch code: GoldBuggyVFinal.ino

Appendix III - GitHub change log of development: Final_Group_Y1_Appendix3_ChangeLog