Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

2E10 Interim Report (Part C)

Group Y1

Buggy Hardware Interface:

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

1. Turning detection and controls.
2. Obstruction detection and controls.
3. 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 left 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.

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.

3. 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, these are either "go" or "stop". These messages are written to the server (Arduino) when a button click of the stop or go button is detected in processing. These inform the Arduino to run the turning and obstruction detection, of to stop the buggy in the same manner as the obstruction detection. The Arduino also sends information back to the supervisor PC in the form of strings such as "Obstruction detected!". This string is written to the client (PC) when the obstruction loop detects a nearby obstruction.

The wheel encoders triggered an interrupt whenever the calculated distance travelled increased by a set amount. This distance was the written to the client as a string also.

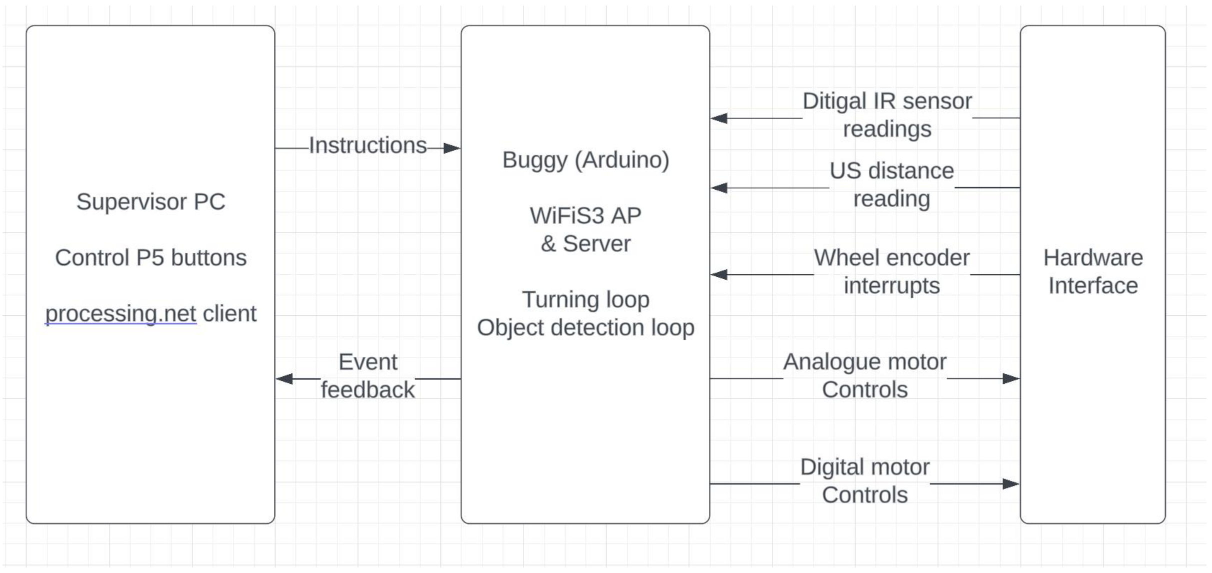

Fig 1. Buggy system model.

Control Flow Algorithms:

Supervisor PC pseudo-code:

```
//import libraries
import controlP5.*;
import processing.net.*

ControlP5 p5;
Button GOButton;
Button STOPButton;

void setup(){
        createButtons(p5, GOButton, STOPButton); //defined control P5 buttons go and stop
        myclient = new Client(); // connect processing client to Arduino server
}

void draw(){
        data = myclient.readString(); //reads incoming events from Arduino
        println(data);
        delay(1000); //print data and wait 1 second
}

public void controlEvent(ControlEvent event){ //called when button click detected in process
```

```
		if(isAControlP5Event){

			if(event = GOButton){
				myclient.write(“go”)
			}

			If(event = STOPButton){
				myclient.write(“stop”)
			}
		}
	}
```

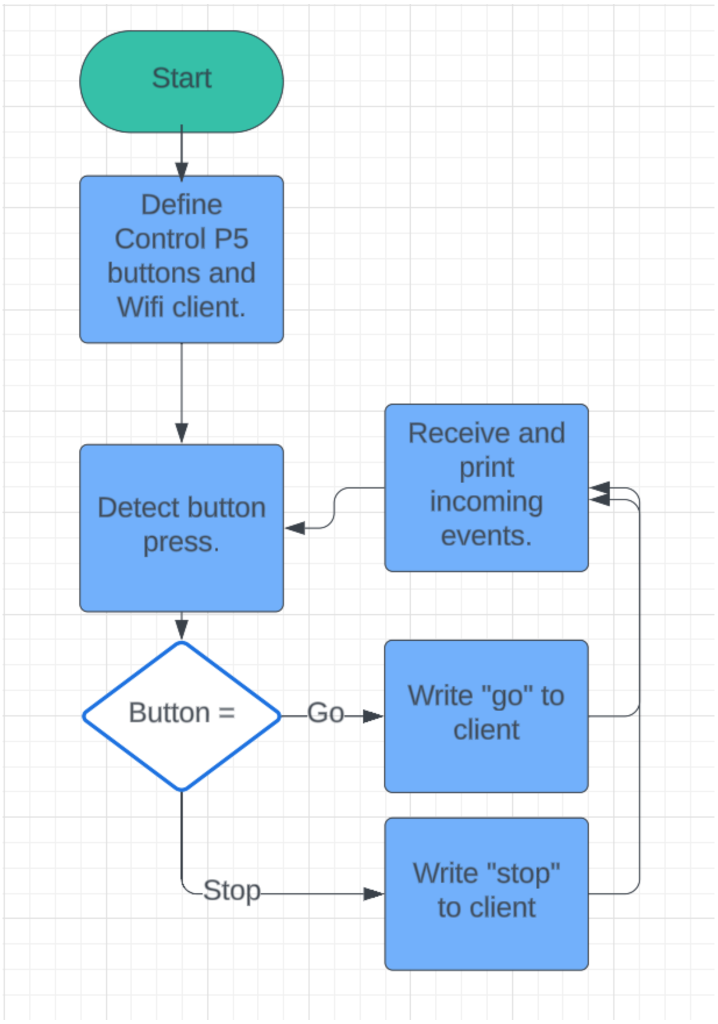

Fig 2. Processing Control Flow

Arduino pseudo-code:

```
#include <WiFiS3.h> //include wifi module

char ssid[] = "Y1_AP"; //define access point and server
char pass[] = "groupY1";
WiFiServer server(5200);


const int RMOTOR1 = 10; //define the pins for the sensors
…..
…
..

void setup(){

        beginSerial();
        beginWifiAP();
        beginServer();
        definePinModes();
        defineWheelInterrupts( function = wUpdate);

        track_colour = readIRSensor();
        startMoving();
}

void loop(){

        client = checkForClientConnection();

        while(client.connected()){

                instruction = receiveMessageFromClient();
                if(instruction == “go”){
                        keepMoving = true;
                }else if(instruction == “stop”){
                        keepMoving  == false
                }

                If(keepMoving){
                        checkForTurns();
                        checkForObstacles();
                        distanceTravelled();
                        cycles++;

                        if(travelled_distance % 10 == 0){
                                client.write(“travelled_distance”);
                        }
```

```
		}else if(!keepMoving){
			stopMoving();
		}
	}
}




Void startMoving(){
	setMotorDirectionAndSpeed();
}

void stopMoving(){
	stopMotors();
}

void checkForTurns(){
	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 checkForObstacles(){
	if(cycles % 15 == 0){
		distance = measureObstacleDistance();
		if(distance < 10){
			stopMoving();
			client.write(“Obstruction detected!”);
		}else{
			keepMovingOrStartMoving();
		}
	}
}

Void wUpdate(){
	wcount = wcount +1;
}

void distanceTravelled(){
	travelled_distance = calculateTravelledDistance(wcount);
}
```

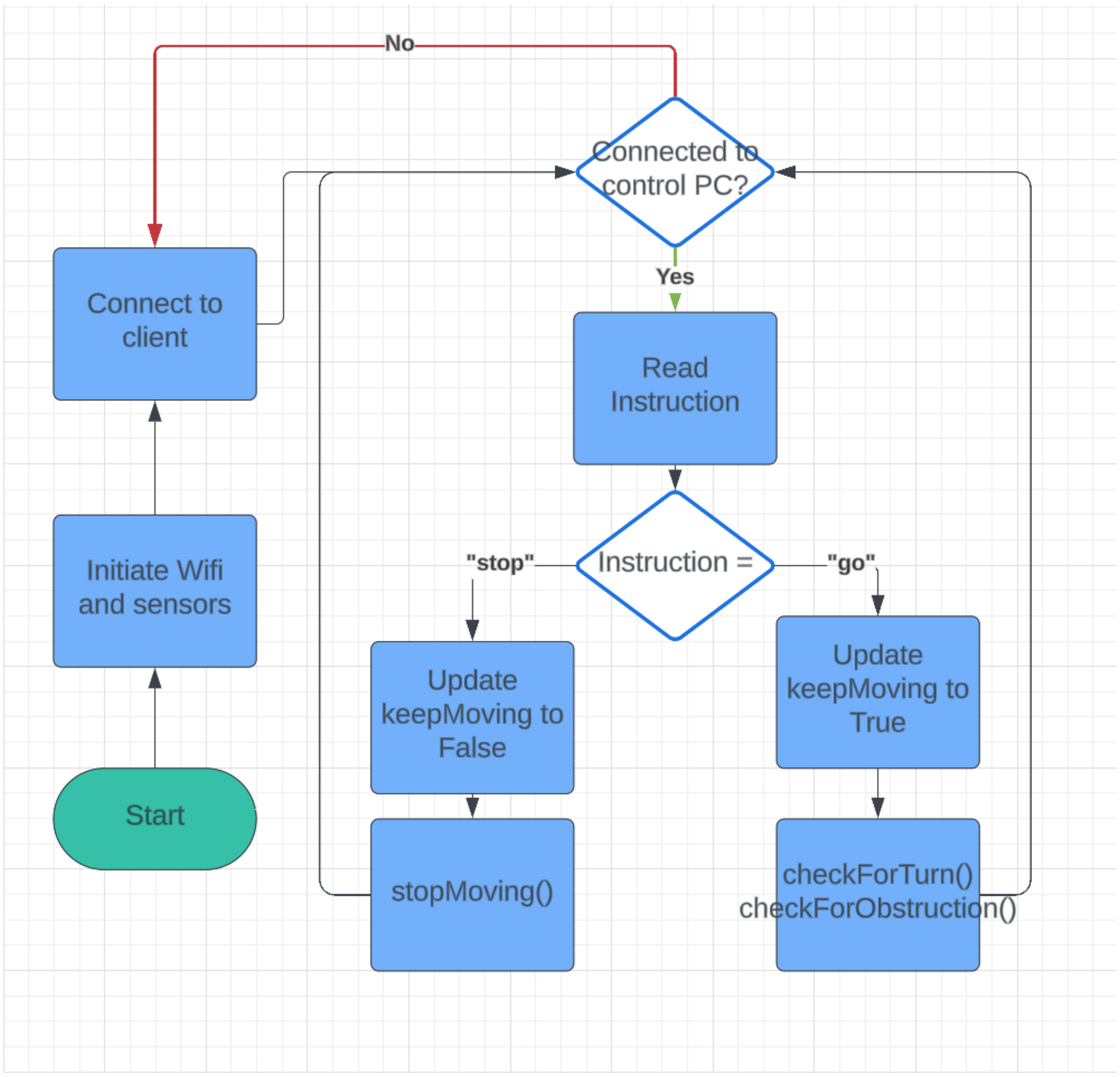

Fig 3. Arduino Control Flow