Uyiosa Imarhiagbe, Zoltán Gere, Albert Offei

Path follower(TM)

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

09.05.2018

Author(s) Title Number of Pages Date	Uyiosa Imarhiagbe, Zoltán Gere, Albert Offei Path follower(TM) 18 pages + 1 appendices May 9, 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialization option	Smart Systems / Devices
Instructor(s)	Keijo Länsikunnas, Senior Lecturer Joseph Hotchkiss, Project Engineer

Project's main objective was to develop an embedded software for Zumo robot control. To achieve this goal elementary theorems had been learned. These basics include but not restricted to numerical representation in 2's complement format, boolean algebra, PWM modulation and PID control. Study of technical manual and electronics datasheets had been necessary. After sufficient knowledge in embedded system programming achieved, the implementation work started with power management to avoid battery ruining. In the next step a basic decision based line following algorithm was developed. The method gave insight how the robot sees the line and how can the current position be measured. To further experiment with the robot control a basic Proportional-Integral-Derivative (PID) control algorithm was developed. Although the code had been finished, it never got fine tuned, only the proportional part is in use with a sufficiently high coefficient. Tests show, by utilizing 4 sensors instead only the two middle, the smoothness of movement can be further increased. Finally a "simple go inside the circle and stay there" algorithm was written for the sumo-style contest.

Keywords	Devices, Smart Systems, Embedded systems, Electronics

Contents

Abbreviation

AC	Alternating current
AD(C)	Analog to digital (converter)
DA(C)	Digital to analog (converter)
DC	Direct current
IDE	Integrated Development Environment
IR	Infrared
NiMH	Nickel-metal hydride
PID	Proportional, Integral, Derivate (control)
PSoC	Programmable System on Chip
PV	Process variable
PWM	Pulse-width modulation
SP	Set-point
UART	Universal asynchronous receiver-transmitter
USB	Universal Serial Bus

1 Introduction

The aim of this project was to study existing technologies and possibly discover new methods to effectively control electronic and robotic systems. The fundamentals of embedded system development was examined and an experimental control software was developed. The main objectives of the embedded software was hardware resource management, overall behavior implementation and predefined task perform. Many basic components of the robot and the micro controller had been explored to achieve these tasks. These modules have been presented in this documentation to provide a complete reference.
In the next chapter all the theoretical knowledge explained, which required to understand the basic operation of the robot's electronic. The third chapter presents the software's actual way of operation. The last chapter brings conclusion and show a possible direction for future development.

2 Theoretical background

First part of this chapter gives insight to theoretical electronic knowledge. The second part presents the electronic modules used in the robot and explains their components' operation.

2.1 Pulse Width Modulation

Pulse Width Modulation (PWM) is a modulation method used to encode information on a carrier signal. PWM is mainly used to empower electronic devices. As the modulated signal alternates between 0 and 1 the device gets an average power instead of continuous output. As a result the devices work in transition between OFF and ON states.

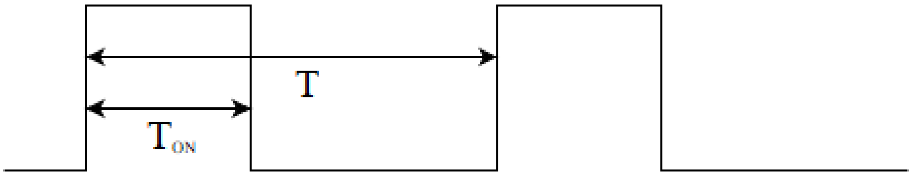

Figure 1: PWM cycle

Duty cycle means the length of ON state (T_{on} in figure) during a full cycle (T in figure). The cycle length or frequency can move on wide spectrum from 1 Hz (1 cycle / second) to 10-100 kHz. (See appendix 1, Frequency)

In this project 1 cycle is exactly 2.56 ms long as 8 bit timer used. Therefore frequency is approximately 390 Hz. 0 value means no movement, brakes are on during the whole cycle. [1]

2.2 PID controller

Proportional-integral-derivative controller or PID controller is a control loop mechanism. The desired position called setpoint (SP). The measured position referred as process variable (PV). The difference of these values give the error value $e(t)$. Based on this error value a new corrected position calculated. Calculation formula has 3 main parts each influencing differently the output.

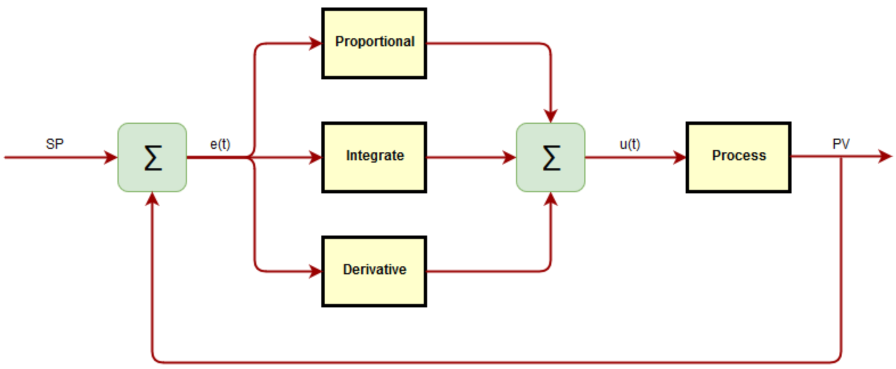

Figure 2: PID controller diagram

The proportional part is contributing linearly, greater current $e(t)$ error value results in greater correction. The integral part collects and integrates past error values. When this integrated error applied, it replaces the error deviation caused by proportional correction. As a result the quickly changing proportional correction replaced by a slowly changing integral correction and the function gets dampened. The derivative part gives a future estimate based on the current changes in the function. It try to zero out the error change rate. Hence the derivative part cannot bring the error to zero.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \tag{1}$$

This mathematical formula contains all three parts. If any part is not to be applied the respective coefficient values should be set to 0. Usually the integral, the derivative or both. [2] [3]

2.3 Cypress CY8 modeling kit

Cypress CY8CKIT is an Arm Cortex M3 based inexpensive prototyping kit. It includes a programmer and debugger modul, making development easier. It is programmed through USB connection. Output terminal is provided on UART port emulated over the USB connection. Software development and device firmware writing performed with PSoC Creator IDE software provided by Cypress, the kits manufacturer.[4]

2.4 Zumo robot

The Pololu Zumo is a small size (less than 10cm) tracked base robot platform. The motors and controller are replaceable allowing customized builds. It includes a steel plate, mounted at the front to protect electronics and to provide capability to push objects. Power source is 4 pieces of AA battery.[5]

2.4.1 Power management

The Zumo robot is powered by 4x 1,2V NiMH batteries. The micro controller runs with 5V and 0V. In order to protect the batteries form too low discharge the voltage is constantly measured. If the voltage drop low the user has to be notified using the led light on the robot.
Because the micro controller operates with clean 5V, and the fully charged batteries reach up to 1.4V each (sum up in 5.6V) the actual voltage is scaled down to 2/3 (See appendix 1, Voltage division rule). This lowered voltage is then directed to micro controllers AD converter unit.[3]

2.4.2 Motor control

Zumo's motors are 6V DC motors.

- Idling: no acceleration and minimum force. Power input: 120mA at 6V.
- Stall: 1.6A / motor. Motor controller restricts current to 1.5A / motor.
- Speed: Run at 400 RPM. The installed gearbox's ratio is 75:1. The top speed is approximately 60 cm/s.

Motor controller unit connects batteries to both motors as instructed by control signals. Motor controller contains H-bridges (DRV 8835). DRV 8835 contains 2 bridges, thus capable to control 2 motors simultaneously.

Electronically there are 3 states the H-bridge can be.

- Forward: direction set to 0, PWM set to 1
- Reverse: direction set to 1, PWM set to 1
- Brake: direction left as it was before braking, PWM set to 0

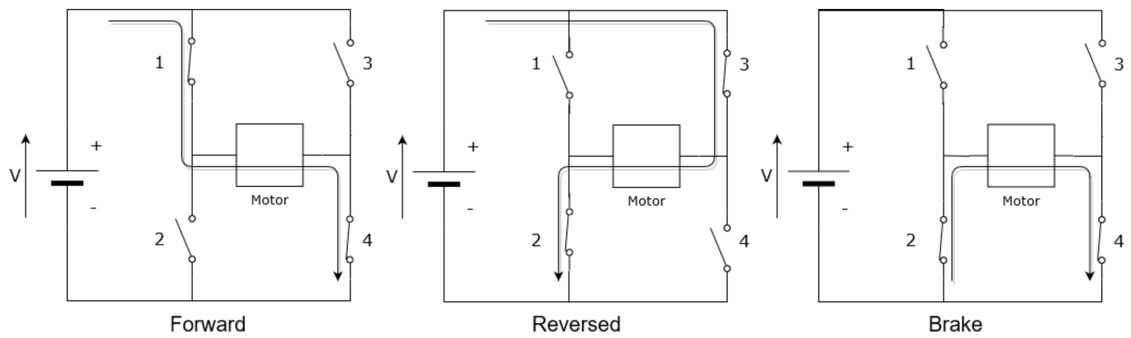

Figure 3: 3 states of motor controller

In brake mode the motor has changed to generator and shorted. In practice this means wheels locked. There is no mode when wheels can freely roll. [3]

2.4.3 Line detection sensors

Zumo has 6 infrared light based sensor positioned at the front. Because IR light is out of visible spectrum, 2 red light led also placed on board. The IR led's light reflected from the surface under the robot back to the sensors. Amount of reflected light is measured using indirect AD conversion. Reflected light activates IR sensitive transistor which close circuit. The current flowing in the circuit is integrated in a capacitor until voltage reach defined level. Time to reach defined level is measured with a micro controller counter. If the surface is white, it has good reflection, so the big current charge the capacitor in short time.
The actual measurement happens in three steps.

- Initial situation: Even when no measurement is happening some current might still flow eventually charging up the capacitor.
- Capacitor discharge: When the micro controller measurement pin gives 5V output, on both sides of the capacitor will be 5V. Consequently the capacitor discharge in 1…2 µs.
- Measurement: Micro controller measurement pin set to input and a timer is started. Capacitor starts to charge again as fast as the IR-sensitive transistor allows it. When capacitor charge reach about 2.5V, voltage at measurement point drops below 2.5V and the pin value turn to 0. Capacitor keeps charging up to 5V. Time to turn from 1

to 0 is measured.

Measurement procedure always takes 1 ms. Different sensors have different sensitivities. Environmental lighting also affects measurement as sun light contains plenty of infrared light.[3]

3 Methods And Materials

In this chapter the process of project development is presented including code explanation and the circumstances of experimenting with Zumo robot.

3.1 The embedded software and mechanics

The section explains algorithms as they control different parts of the robot, introduced in the previous chapter. Movement controlling formulas and program flows also explained.

3.1.1 Voltage measurement

Library for battery management is named <ADC_Battery>. It is part of micro controller standard library and the source can be found in ’codegentemp’ folder. Prior to use, it has to be initialized with ADC_Battery_Start() command. Actual measuring is not instantaneous. The process is started with the ADC_Battery_StartConvert() command.
Wait to finish measurement is achieved in one step: ADC_Battery_IsEndConversion(ADC_Battery_WAIT_FOR_RESULT). The actual value is queried with ADC_Battery_GetResult16() instruction. As the micro controller operates with 5V, measurement range is 0V - 5V. The AD converter is 12 bits unsigned therefore the return value is in 0 - 4095 interval. The formula of actual voltage:

$$volts = \frac{ADcode}{4095} * 5 * 1.5 \quad (2)$$

. As mentioned in Chapter 2's Zumo robot section, battery voltage scaled down to 2/3. The trailing multiplication by 1.5 in the formula compensates this scale down.

3.1.2 Decision based line following

Decision based line following algorithm is recommended to develop first. It is straightforward to implement and it can provide insight how the sensors see the path. Also help understand the characteristic and dynamic of the motors, how the small changes affect the robot's overall movement.
This method use two state, black and white values only. If the threshold parameters set accurately the method can operate in various light conditions.
This algorithm utilizes all 6 sensors to navigate along the path. The middle two sensors used in two ways. If both read black value, the robot motor speeds are increased toward equal values, thus bringing the robot to straight. If one sensor measure black but the other one white, it is regarded as small deviation from path and minor speed adjustment applied. The two mid-end sensors detect greater deviation from path and utilize larger corrections. The two end sensors used for path intersection detection only.
Crossing lines detected in two step by recognizing black-to-white transitions.

- In first step, if both sensors report black value, Line_found flag is set.
- Secondly, if sensors read white values and Line_found flag is 1 means the robot just passed from black to white. Consequently line counter variable increased and the flag set to 0.

Experiments has shown, if-then control can be used to 'convert' smooth path to polygonal track. If the curvatures regarded as rough polygons, the robot movement becomes straight run with sudden direction changes resulting in faster method, than what PID control can provide.

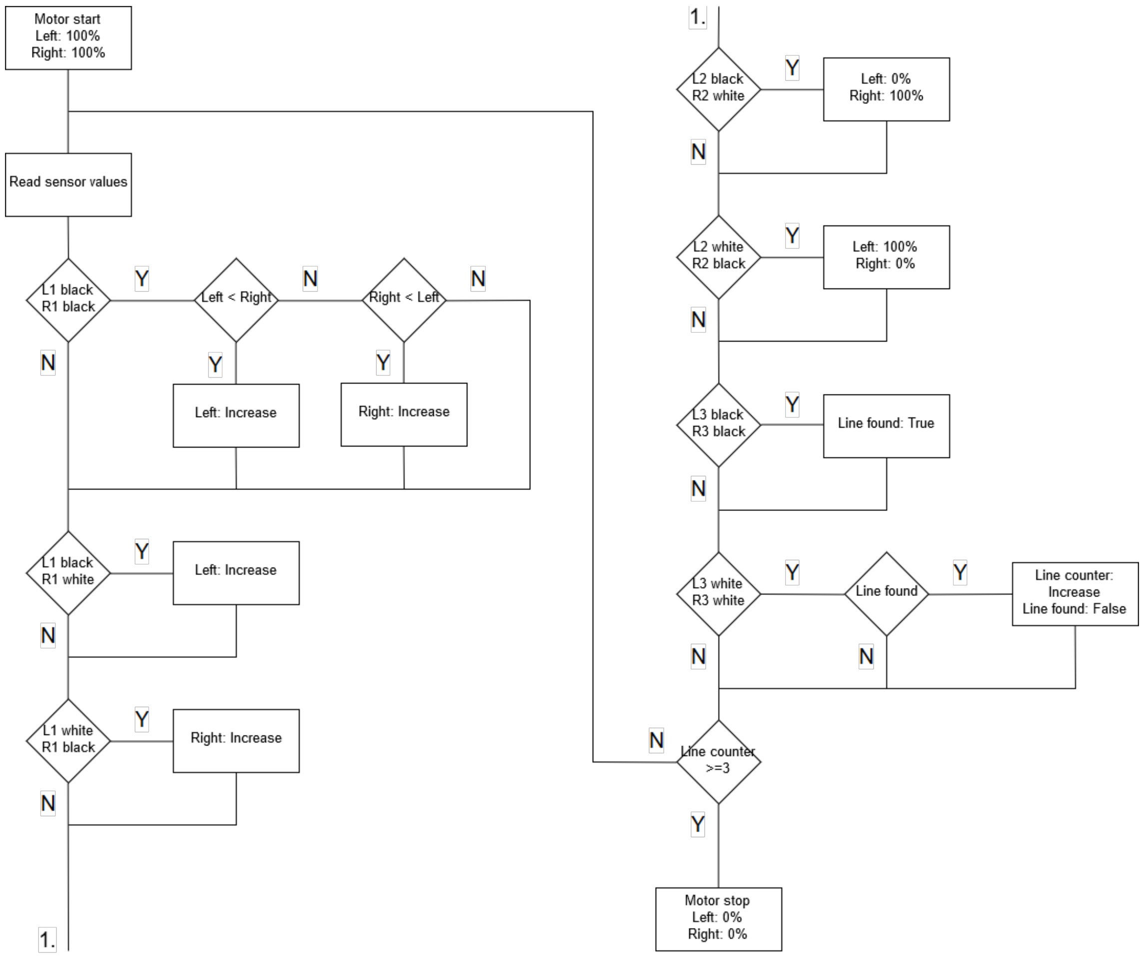

Figure 4: Line following mechanism

3.1.3 PID control and error calculation

This method can provide the most accurate line following. It results in smooth although not the fastest run.

First an error rate must be calculated. It is based on the two middle sensor's average value.

$$ErrorRate = \frac{BlackValue - AverageReading}{BlackValue - WhiteValue} \tag{3}$$

The error rate should be between 0 and 1. Because $BlackValue$ and $WhiteValue$ contains some tolerance (to cover lighting circumstances), error rate can be outside of these boundaries. The direction of deviation also included in the value. Negative sign means deviation to the left side, positive means to the right.

Next the integral and derivative parts are calculated.

$$Integral = Integral + ErrorRate * DeltaTime \tag{4}$$

$$Derivative = \frac{Error - PreviousError}{DeltaTime} \tag{5}$$

In following step, the $ErrorRate$, $Integral$ and $Derivative$ values are multiplied by their respective coefficients and summed up. The absolute value of result is calculated. Finally, based on the sign of error rate (indicates direction) correction is applied to opposite side motor speed. (If robot is on left side of track the right motor must brake.)

3.1.4 Sharp turn calculation

If $ErrorRate$ drops to zero, it means the robot lost track completely. In this situation it should find the way back to the path. To achieve this the two end sensor values are continuously monitored. If any of them reads $BlackValue$, the time is recorded. When the track is lost, sharpturn mode turned on. In sharpturn mode PID control is disabled and more direct control utilized.

The robot can not leave the track without reading on any side $BlackValue$, therefore one of the time stamps is indicating where should the robot look for the path.

When sharpturn mode is active, the robot motors are programmed with maximal speed, but opposite directions. This way the robot start spinning around its center axis. When $ErrorRate$ becomes again 1 sharpturn mode turned off and control is returned for PID controller.

3.1.5 Motor speed programming

Desired speed value is stored in percentage. 100% means the preset travel speed value. It is also signed value. Positive sign means forward, while negative sign means reversed direction movement.

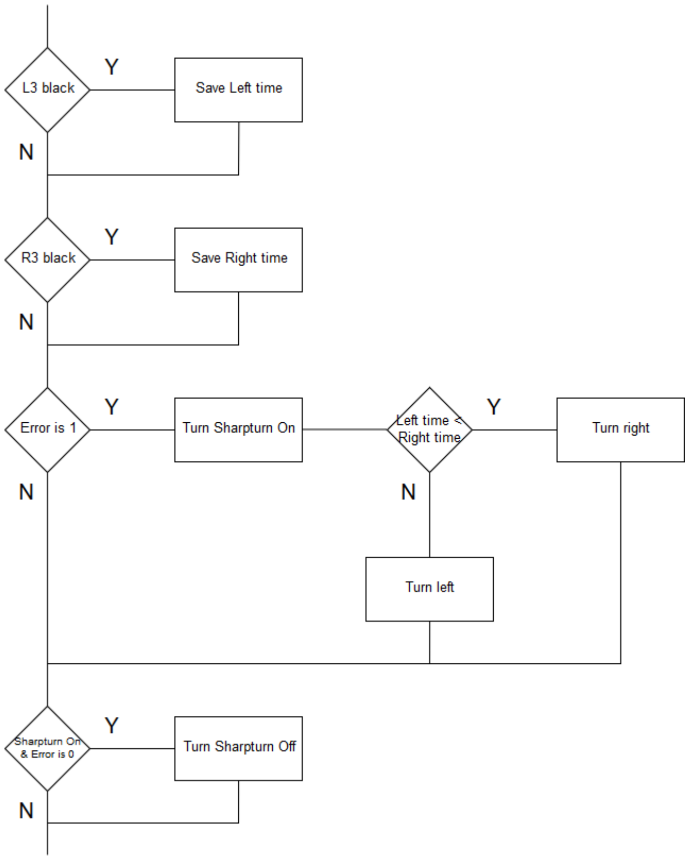

Figure 5: The sharp turning mechanism

$$ActualSpeed = \frac{DesiredSpeed * TravelSpeed}{100} \tag{6}$$

After the calculations of left and right side speeds, the absolute values are programmed into the PWM controller.

Finally the motor directions programmed according to the sign of desired speed.

3.2 Timing

The work phases distribution over time is presented here.

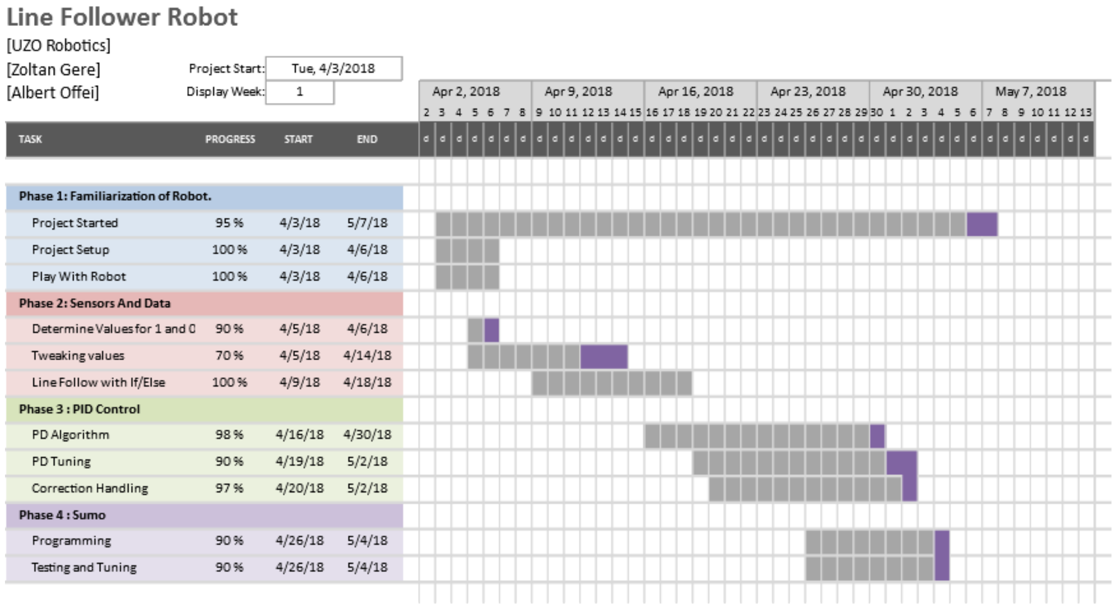

Figure 6: Development timeline

3.2.1 Familiarisation Of Robot

3.2.2 Project Setup

PSoC creator software was downloaded and installed. Github was used as version control. To achieve this, one member of the group created a Github repository and other members were added as contributors. Considering the time frame for the project, a library was provided which had the hardware and the various components set up. The code needed to follow the line and to participate in the Sumo fight was then programmed using this library.

3.2.3 Play With Robot

We experimented with the robot forward movement as a first step. The motor_forward function provided by the library, handles this. The code snippet bellow shows the motor_forward function.

```
void motor_forward(uint8 speed,uint32 delay)
{
```

```
    MotorDirLeft_Write(0);     // set LeftMotor forward mode
    MotorDirRight_Write(0);    // set RightMotor forward mode
    PWM_WriteCompare1(speed);
    PWM_WriteCompare2(speed);
    CyDelay(delay);
}
```

Listing 1: Robot Forward Move Function

As shown in code snippet , the motor_forward function takes uint8 (unsinged 8bit Integer) value of 100 which represents the speed value as the first parameter and 2000 as the delay.

```
motor_forward(100,2000);      // moving forward
motor_turn(200,50,2000);      // turn
motor_turn(50,200,2000);      // turn
motor_backward(100,2000);     // moving backward
```

Listing 2: Robot Forward Move Code

3.2.4 Sensors And Data

The robot uses six reflective sensors. The values provided by the sensors are higher when on black, around 23000 and about 5000 when on white. As the robot nears black, the values increases and vice versa. For the Sumo fight, an ultrasonic sensor was used to measure distance. The ultrasonic sensor served as the eye of the robot to determine the proximity of other robots. Reflectance_start() function from the library was used to start the sensors.

```
void reflectance_start()
{
    IR_led_Write(1);
    //sensor_isr_StartEx(sensor_isr_handler);
    Timer_R1_Start();
    Timer_R2_Start();
    Timer_R3_Start();
    Timer_L3_Start();
    Timer_L2_Start();
    Timer_L1_Start();
    refl_init = true;
    Systick_Start();
}
```

Listing 3: Reflectance

3.2.5 Tweaking Values

```
reflectance_set_threshold(9000, 9000, 11000, 11000, 9000,
    9000); // set center sensor threshold to 11000 and others
    to 9000
```

Listing 4: Reflectance Threshold

The reflectance_set_threshold() function above takes six parameters which are the threshold above which a sensor output is considered a logic 1 and bellow a logic 0. During this phase of the project, we played with several possible values. To do this, we had the robot connected to the laptop without motor running and moved the robot on the black and white to see how the values changed.

As the robot had been moved over the black line we get the above sensor values. The

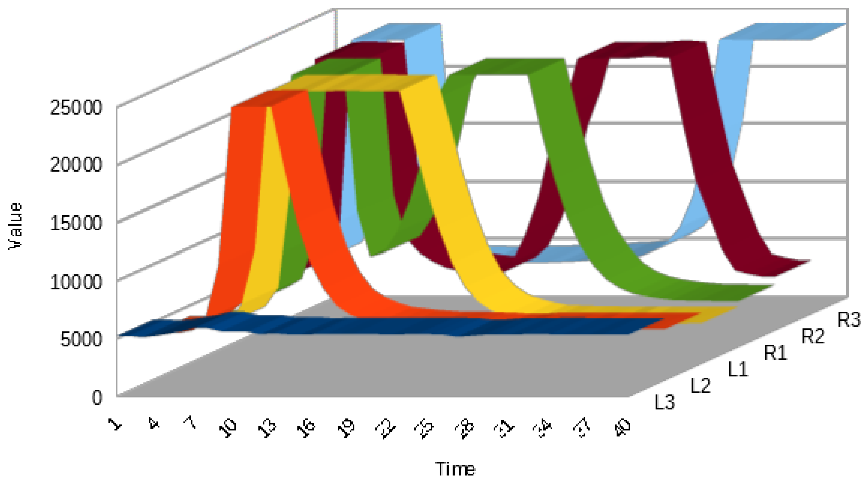

Figure 7: Value set read by the sensors

peak values form the black line and the low 'valley' values represent the white background.

3.2.6 Line Follow With If/Else

```
//IF AND ELSE STATEMENTS WITH DIGITAL VALUES OF 4 MIDDLE
    SENSORS

    if (!dig.l2 &&dig.l1 && dig.r1 && !dig.r2) // (0110)all
        clear condition
    {
        motor_forward(255, 10);
    }

    else if (dig.l2 && dig.l1 && !dig.r1 && !dig.r2) //
        (1100)smaller turn here to the left
```

```
    {
         motor_turn(0,180,10);
    }

    else if (dig.l2  && !dig.l1 && !dig.r1 && !dig.r2 ) // (1000) faster turn to the left
    {
       // motor_turn(0,180,10);
         MotorDirRight_Write(0);
        MotorDirLeft_Write(1);
        if (dig.l1 && dig.r1)
        {
            MotorDirRight_Write(0);
            MotorDirLeft_Write(0);
            motor_forward(255,10);
        }

    }

    else if (!dig.l2 && !dig.l1 && dig.r1 && dig.r2) // (0011)smaller turn here to the right
    {

        motor_turn(180,0,10);

    }

    else if (!dig.l2  && !dig.l1 && !dig.r1 && dig.r2 ) // (0001) faster turn to the right
    {

        MotorDirRight_Write(1);
        MotorDirLeft_Write(0);
        if (dig.l1 && dig.r1)
        {
            MotorDirRight_Write(0);
            MotorDirLeft_Write(0);
            motor_forward(255,10);
        }
    }
```

Listing 5: If Else Line Follow

Using the above code, the robot was able to follow the line considerably well albeit not smoothly enough. The If/Else code above, takes into account the digital values that the sensors output which was determined during our value tweaking phase. From the first If statement, the motor_forward function takes the maximum speed value of 255 and a delay of 10 milliseconds, if both middle sensors are on black and hence has a value of

1. After 10 milliseconds, the condition is checked again and moves to the next code if condition has changed. The first else if condition statement, checks if sensor l2 and l1 can both see black and if r2 and r1 both see white, in that case, we give the left motor a speed of 0 and the right motor a speed of 180 cause the robot to turn left. A check was performed again after 10milliseconds. This approach worked for simple turns but failed at sharper turns. We found that one way to work the sharper turns was to make use of the motor_backwards function of the library.

```
void motor_turn(uint8 l_speed, uint8 r_speed, uint32 delay)
{
    PWM_WriteCompare1(l_speed);
    PWM_WriteCompare2(r_speed);
    CyDelay(delay);
}


/**
* @brief    Moving motors backward
* @details  setting backward mode to each motors and gives same speed to each side of PWM
* @param    uint8 speed : speed value
* @param    uint32 delay : delay time
*/
void motor_backward(uint8 speed,uint32 delay)
{
    MotorDirLeft_Write(1);      // set LeftMotor backward mode
    MotorDirRight_Write(1);     // set RightMotor backward mode
    PWM_WriteCompare1(speed);
    PWM_WriteCompare2(speed);
    CyDelay(delay);
}
```

Listing 6: motor_turn and motor_backwards

The if else line follower implemented the MotorDirLeft_Write(1) to cause the left motor to move backward whiles the right motor moved forwards making it possible to make a tighter left turn.

3.2.7 PD Control

A simple proportional and derivative (PD) control was tested on the robot. The code is shown bellow. The robot was able to follow the line this way but not as smoothly as expected. The PD control required a lot of try and error by observing the robots behaviour when kpValue and kdValue were varied. A more complex PD control was later developed which run much smoothly.

```
//PD Control

        setPoint = 14000; // value to be compared to, ref.l1
            close to edge of line

        kpValue = 1.3; // modifiable

        kdValue = 0.5; // modifiable

        error = setPoint - ref.l1; //error calculation in
           relation to ref.l1

        //previousError = error; //differential error

        correction = kpValue * error; //+ kdValue * (error-
           previousError); //correction to be used

        rightMotorSpeed = rightSpeed - correction;
        leftMotorSpeed = leftSpeed + correction;


        if (rightMotorSpeed > rightSpeed) rightMotorSpeed =
           rightSpeed;

        if (leftMotorSpeed > leftSpeed) leftMotorSpeed =
           leftSpeed;
```

Listing 7: Simple PD Control

3.2.8 Sumo Fight Programming and Tuning

The sumo fight code was a basic algorithm that took into consideration distance of other robots and a decision to move towards the robots and attempt to push them out of the ring. A simple algorithm was written to this effect. The Sumo code is shown bellow.

```
int distance;

    int seek()
    {
         motor_turn(0,155,0);            //Keep turning in
            a circle until
        return 0;
    }
```

```
int attack(int distance, int digl3, int digr3)  // function to attack objects at a certain distance
{
    if (distance <= 35)
    {
        motor_forward(255,1);       //if an object is about 35cm away or less, move towards it
    }

    else

    {
        seek();                     //object more than 35 cm away, ignore and keep searching
    }

    if (digl3 || digr3)
    {
        motor_backward(200,0);      //if sensors see black, move backwards and turn
        motor_turn(0,200,0);
    }

    return 0;
}
```

Listing 8: Sumo

4 Conclusion

The objective of this project was to develop controlling software for Zumo robot. It has been accomplished, as the robot is capable of following the path up to the finish line. During development and race the robot worked well and batteries have been spared.

The if-then method works well. The PID controller also works on algorithm level. It can not be used yet, as fine tuning of parameters is necessary. But if the robot is intended for racing, the if-then approach should be further developed and tuned, because PID controller gives smoother but slower result.

Development might be considered successful on educational level. Developers acquired basic electronic, device, programming and controlling knowledge. These can be used in future researches and developments.

Bibliography

1 various editors. Pulse Width Modulation;. Available from: https://en.wikipedia.org/wiki/PID_controller [cited April 23, 2018].

2 various editors. PID controller;. Available from: https://en.wikipedia.org/wiki/PID_controller [cited 24.04.2018].

3 Leinonen T. Black board pictures;.

4 Corp CS. CY8CKIT-059 PSoC® 5LP Prototyping Kit With Onboard Programmer and Debugger;. Available from: http://www.cypress.com/documentation/development-kitsboards/cy8ckit-059-psoc-5lp-prototyping-kit-onboard-programmer-and [cited April 24, 2018].

5 Pololu. Zumo robots and accessories;. Available from: https://www.pololu.com/category/129/zumo-robots-and-accessories [cited April 24, 2018].

1 Physics

1.1 Frequency

Frequency means for periodical functions (e.g. signals) the number of periods completed in 1 second. Unit of frequency is Hertz (Hz). For example 1 period in 1 second is 1 Hz. 10 period in 1 second is 10 Hz.

1.2 Infrared light

Infrared light (IR) is 700nm to 1mm section of light spectrum. The wavelength of IR is longer than that visible by human eye. This invisibility gives IR light wide range of purpose.

1.3 Electricity

1.3.1 Voltage division rule

Voltage division is a method to produce output voltage, which is fraction of the input voltage. If 2 resistor connected in series, the input voltage is distributed between them. The output voltage can taken from between the resistors.

$$V_o ut = \frac{R_2}{R_1 + R_2} * V_i n \tag{7}$$

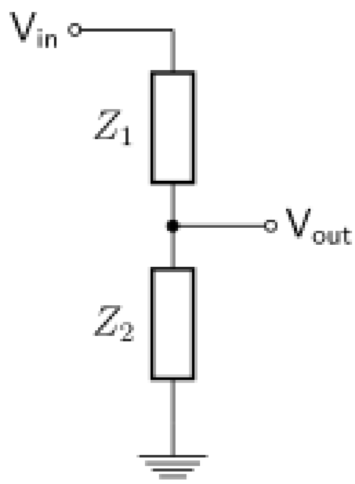

Figure 8: Voltage divider circuit