Robo 3.1
An Autonomous Wall Following Robot

Samarth Mehta*
Department of Electrical and Computer Engineering
Cleveland State University
Cleveland, Ohio 44115

May 1, 2008

* Email address: s.a.mehta69@csuohio.edu. This work was primarily supported by Provost of Cleveland State University through Honors Department under the program called REU (Research Experience for Undergraduates) in Summer 2007.
Abstract

This report presents a wall following autonomous mobile robot—Robo 3.1. It follows a wall on its left. Robo 3.1 used a PIC 18F4520 microcontroller as its brain. PIC 18F4520 receives input from Ultrasonic Distance Meter (UDM). Some computations are performed on this input and a control signal is generated to control the robot’s position. This control signal is generated through a PD controller, which is implemented in the microcontroller. First, simulations were performed in Matlab to design a PD controller and to understand the overall picture of the robot’s behavior. Afterwards, microcontroller was programmed in C language using a CCS compiler. The robot was fine tuned during the tests. The test results are illustrated.

1. Introduction

Autonomous robots are electromechanical devices that are programmed to achieve several goals. They are involved in a few tasks such as moving and lifting objects, gathering information related to temperature and humidity, and follow walls. From a system’s engineering point of view, a well designed autonomous robot has to be adaptable enough to control its actions. In addition, it needs to perform desired tasks precisely and accurately.

The overall objective of this project is to follow a wall on its left and maintain a constant distance with it. (Left sides and right sides are considered from robot’s point of view unless it is specified.) In general, the robotic chassis are designed carefully, so that they are balanced. After loading this chassis with the battery packs and other electronics, the chassis become unbalanced. In addition, the plane on which the robot is moving could also be uneven. These result in a veering of the robot. This may cause the robot either to run into a wall, or go farther away from the wall. Therefore, a decent control technique is required to tackle this problem. In summer 2007, a team of six undergraduate students developed a project—Robotic Swarms [8]—under the supervision of a graduate student and an associate professor at Cleveland State University. One of the tasks of this project was to implement a wall following capability in to the robots. A closed loop control technique was used to follow a wall that had large response time to reach to the set value. Therefore, there was a need to develop a system that can improve the response time.

1.1. Literature Search

In order to follow a wall and navigate through small pathways, image processing techniques can be used. This technique requires cameras that can be expensive. Reference [9] suggests using two switches on the left of the robot to follow left wall and a bumper switch to detect an obstruction in front. A genetic algorithm using Fuzzy Logic Controller is developed in [7]. A multivariable control and ultrasonic range finders for the measurement of a distance are being used for wall tracking robot in [2]. A Fuzzy Logic Controller is used to drive a robot parallel to the wall with the help of ultrasonic sensors in [13].
1.2. Summary of Sections

Section 2 discusses the choice of controller and mathematical model along with the simulations in Matlab. Section 3 explains hardware and software for motors, Ultrasonic Distance Meter and PD Controller in detail. Section 5 gives the conclusion based on the observations and results.

2. Controller, Mathematical Model and Simulations

2.1. Controller

There are several techniques that can be used to control mobile robots. One of the techniques is an open loop control. In the open loop control system of a robot, the inputs such as speed, distance, a path between the start and end points are calculated beforehand. This technique does not handle disturbances or correct one of the parameters that could go wrong. Other technique that can be used is a closed loop control. The closed loop control system can compensate for the errors occur in real time since the inputs are based on the actual conditions and not predefined by the assumptions. In order to have accurate and precise controlled actions of a robot a closed loop system is preferred over an open loop system. There are various types of controllers that can be used in a closed loop control system. The basic controllers that are used widely are Proportional (P) controller, Proportional Integral (PI) controller, Proportional Derivative (PD) controller, and Proportional Integral Derivative (PID) controller. For a wall following robot, PD controller turns out to be a fairly decent controller, which will be discussed in Section 2.3 while discussing simulations of PD controller.

2.2. Mathematical Model

Different designs such as legged or wheeled robots have different mathematical models. Furthermore, wheeled robots have several types such as synchronous drive, Omni directional and differential drive [3]. The differential drive method is simple to implement and sharp turns can be made. Therefore, differential drive technique is used for this project.

The differential drive robots with wall following robots have a number of applications. For example: building a map of an unknown area, manufacturing, rescue services and various military applications, and collecting data and monitoring various parameters such as temperature and humidity.

In general, robot is a non-linear system that can be understood from the following equations [2].

\[ x = \frac{v_l + v_r}{2} \cos \theta \] (1)
\[ \theta = \frac{v_r + v_l}{L} \]  
\[ y = \frac{v_l + v_r}{2} \sin \theta \]

The following list gives the meaning of the symbols used in the above equations.

- \( v_l \) = left wheel linear velocity
- \( v_r \) = right wheel linear velocity
- \( L \) = distance between two wheels
- \( x \) = horizontal distance
- \( \dot{x} \) = rate of change in horizontal position
- \( y \) = vertical distance
- \( \dot{y} \) = rate of change in vertical position
- \( \theta \) = angular position of a robot with respect to \( x \)
- \( \dot{\theta} \) = rate of change in angular position

One can see from the above equations, that the robot has a non-linear system because of cosine and sine terms. In this project, robot needs to maintain a constant distance from a wall, which is a horizontal distance. In this project, a wall following robot does not need to control vertical position. Therefore, there is no need to use (3).

### 2.3. Matlab Simulations

Matlab simulations are generally used to get assistance in understanding an overall picture of the problem and to find a decent solution to the problem. It is clear from (1), (2) and (3) that Robo 3.1 is a non-linear system. Therefore, checking a stability of the robot is little bit complicated. Hence, a simple code in m-file was written which can be viewed in the Appendix A. The simulations are given in Figure 1 and Figure 2. The first figure illustrates the graph for X position vs. Time when \( \Delta t \) is 20 ms. \( \Delta t \) is a time interval between two sets of UDM readings. Each set of UDM reading consists of three ultrasonic readings (left_front, left_back and front distance). The second figure illustrates the graph for X position vs. Time when \( \Delta t \) is 60 ms. The reference value is chosen as 20 cm.
Figure 1: X position vs. Time when $\Delta t = 20$ ms

Figure 2: X position vs. Time when $\Delta t = 60$ ms
Figure 1 proves that gains chosen for PD controller for $\Delta t = 20$ ms are correct. On the other hand, same gains are improper when $\Delta t = 60$ ms as shown in Figure 2. Ultrasonic Distance Meter may take approximately 450 $\mu$s to 60 ms. Even if ultrasonic readings are taken continuously without waiting for 10 to 100 $\mu$s, $\Delta t$ will be about 60 ms in worst case. Tuning a controller based on such wide variations is very difficult. However, it can be concluded from first figure that if $\Delta t$ is quite small and if it does not vary with large amount than PD controller can perform appropriately.

3. Hardware and Software

3.1. Construction of a robot

Figure 3 illustrates the hardware block diagram of Robo 3.1. As it can be seen clearly from Figure 3, it has two wheels on either side of its chassis. Each of these wheels has a separate DC motor. These motors run independently from each other with the help of PWM signals generated by the PIC18F4520 microcontroller, and a driver IC SN754410. Moreover, a caster wheel is used in front and back to balance the entire chassis.

Robo 3.1 measures the distance from a wall on its left and protects itself from an obstruction in front. Therefore, it has three ultrasonic sensors. Two of the ultrasonic sensors are on the left side to aid in following a wall and one of the ultrasonic sensors is in front—primarily used to detect an obstruction.

In order to have less noise involved from motors and to provide enough energy to the electronics on board, two power supplies are used. One power supply is dedicated to motors and the other power supply is dedicated to the circuit board, which is mounted on Robo 3.1.
3.2. Electronics

PIC 16F877 was used in the class for labs, but PIC18F4520 is chosen over PIC16F877. They both have 8 bit processor, but from Table 1 it can be seen that PIC18F4520 has more resources as compared to PIC16F877. PIC18F4520 can operate faster than PIC16F877, because PIC18F4520 has operating frequency range up to 40 MHz. On the other hand, PIC16F877 can operate up to 20 MHz. PIC18F4520 has more SRAM and Program memory (Flash) and has one extra timer as compared to PIC16F877. PIC18F4520 also has three 16 bit timers as opposed to only one 16 bit timer in PIC16F877.
### 3.2.1. Ultrasonic Distance Meter (UDM)

#### 3.2.1.1. Introduction of an Ultrasonic Sensor

Robo 3.1 has three ultrasonic sensors. Two of the ultrasonic sensors are used to locate walls on its left side and detect obstacles in the front. The ultrasonic distance sensor provides precise, non-contact distance measurements. It transmits an ultrasonic burst to detect any obstacle and outputs a pulse. The output pulse width corresponds to the time required for the burst echo to return to the sensor. The distance between the robot and a wall can be determined by multiplying the pulse width and the velocity of the ultrasonic burst. The ultrasonic sensor (PING) by Parallax Inc. has a male three-pin header used for +5V supply (Vdd), ground (Vss), and signal as shown in Figure 4. Since there is only one pin for signal, the input pulse transmitted and the output pulse received at the same pin.

<table>
<thead>
<tr>
<th>Peripherals</th>
<th>PIC18F4520</th>
<th>PIC16F877</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating Frequency</td>
<td>DC-40 MHz</td>
<td>DC-20 MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32k x 8</td>
<td>8k x 14</td>
</tr>
<tr>
<td>SRAM</td>
<td>1536</td>
<td>368</td>
</tr>
<tr>
<td>EEPROM</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>I/O</td>
<td>36</td>
<td>33</td>
</tr>
<tr>
<td>A/D Channels</td>
<td>13</td>
<td>8</td>
</tr>
<tr>
<td>CCP/ECCP</td>
<td>1/1</td>
<td>2/0</td>
</tr>
<tr>
<td>SPI/I2C</td>
<td>Y/Y</td>
<td>Y/Y</td>
</tr>
<tr>
<td>EUSART</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Timers 8/16 bits</td>
<td>1/3</td>
<td>2/1</td>
</tr>
</tbody>
</table>

Table 1: Comparison between PIC18F4520 and PIC 16F877 [4], [5]

![Ultrasonic Sensor Pin Configuration](image)
3.2.1.2. Theory of Operation [6]

PIC18F4520 by Microchip Technology Inc. (host) provides input pulse of the width of about 10 µs to the ultrasonic sensor on its SIG pin. The ultrasonic sensor transmits a burst signal of 40 kHz for about 200 µs. The burst signal travels through air at the velocity of 1130 ft/s. Once the input pulse of 10 µs is sent to the sensor, the sensor waits for about 750 µs. After 750 µs, the SIG pin becomes output pin and becomes high. It remains high until the burst signal hits an object and returns to the sensor. This pulse width provides the time for the ultrasonic wave to travel between the robot and the object. The timing diagram is as shown in Figure 5.

![Timing Waveforms for Ultrasonic Sensors](image)

- $t_{\text{OUT}}$: 2 µS (min), 5 µS typical
- $t_{\text{HOLDOFF}}$: 750 µS
- $t_{\text{BURST}}$: 200 µS @ 40 kHz
- $t_{\text{IN-MIN}}$: 115 µS
- $t_{\text{IN-MAX}}$: 18.5 mS

Figure 5: Timing Waveforms for Ultrasonic Sensors [6]
For the convenience of testing, three seven segment LED displays were used to display a distance between an object and the robot. This technique was used initially to test the ultrasonic sensors. Later on, LCD was used on the robot for troubleshooting purposes as it requires less amount of wiring on the circuit board. Moreover, meaningful sentences can be displayed on LCD that helps in troubleshooting and verifying different parameters. A code for LCD is given in Appendix A and it is fairly easy to understand, so it is not explained in detail in this report. Also, LCD code was not used during the actual demonstration as it keeps the processor busy and provides less time for the controller. The connection for the three seven segment LED display is shown in Figure 6.

![Connections for seven segment LED Displays](image)

Figure 6: Connections for seven segment LED Displays [10]

### 3.2.1.3. Software for Ultrasonic Distance Meter

The program in Appendix A demonstrates the use of ultrasonic sensor using PIC18F4520. It also displays a distance on three seven-segment LED displays. The input pulse (low-high-low) provided by the PIC using output_low() and output_high() functions to trigger the ultrasonic sensor. After the trigger pulse falls, the sensor will wait about 200 µs before transmitting the ultrasonic burst and waits for about 750 µs before converting the SIG pin into an output pin. This will provide enough time for the PIC18F4520 microcontroller to load and prepare for the next instruction. A while loop is
used to determine when to start the timer. The program control goes in the while loop and remains in it to check if output pin of ultrasonic sensor goes high. As soon as signal goes high, timer1 starts from 0. The Timer 1 interrupt is used to check an overflow. Once Timer 1 has started, program control goes in another while loop and stays there until output pin goes low. As soon as output signal goes low, the Timer 1 value is received using get_timer1() function. The raw time value from Timer 1 must be scaled due to overflow of Timer 1 and the overhead time consumed by the instructions. Also, the total time needs to be divided by two to remove the return trip of the echo pulse.

display_number() function is used for displaying distance on the seven-segment display. To save pins on the microcontroller, multiplexing was used. In other words, each digit was displayed for 2 ms and this was done continuously in a loop. Due to the persistence of vision, display looks steady and all three digits can be viewed together.

3.2.2. Motors and driver circuit

3.2.2.1. Hardware for Motors

The motors used for Robo 3.1 are DC motors. These DC motors are rated at 5 V. Texas Instruments’ SN754410 Quadruple Half-H Driver is used to drive two DC motors. Figure 7 illustrates the connection for the motor drivers and motors.

Figure 7: Motor Connections for SN754410 (output diodes are internal.) [12]
As shown in Figure 7, SN754410 has four buffers and two enable pins. Each enable pin controls two buffers. These enable pins are controlled by the PIC18F4520 microcontroller. When these enable pins are high, the buffers turn on and when enable pins are low, the buffers turn off. By disabling the enable pins, the motors can be turned off. Pin 2 of Buffer 1 and Pin 7 of Buffer 2 in SN754410 require PWM and complimentary PWM signals, respectively. Similarly, Pin 10 of Buffer 3 and Pin 15 of Buffer 4 require PWM and complimentary PWM signals, respectively. The PIC18F4520 is capable of providing only two PWM signals that are not complimentary of each other. Therefore, MC74HC00SN NAND gates were used as inverters to provide complimentary PWM signals. IC 7404 can be used instead of IC 7400. IC 7400 was used because of its availability in the lab.

Motors are connected as shown on the left in Figure 7. Motors can also be connected as shown on the right in the above figure, but they can run only in one direction. Pin 16 of PIC18F4520 is connected to Pin 2 of the driver IC. Also, the PWM signal on Pin 2 of the driver IC is inverted and connected to Pin 7. Pin 17 of PIC18F4520 was similarly connected to Pin 15 of the driver IC and inverted PWM signal to the Pin 10 of the driver IC.

3.2.2.2. Software for Motors

The program in Appendix A gives an example of PWM code that was tested to drive two motors. First, CCP1 and CCP2 are configured for PWM by using setup_ccp1(CCP_PWM) and setup_ccp2(CCP_PWM). Then, Timer 2 was configured by setup_timer_2(T2_DIV_BY_16, 255, 1). Here, the first argument is the mode. T2_DIV_BY_16 means divide the frequency by 16. Mode can be selected as 1, 4 or 16. The second argument indicates the period value. This number is used to determine the PWM period. Timer 2 counts up to the number defined in this second argument. The last argument is the postscale that determines how many timer overflows are allowed before an interrupt occurs. In order to set on time for the PWM signal, set_pwm1_duty (value) or set_pwm2_duty (value) can be used. Once Timer 2 reaches the number equal to the argument, the PWM output switches from high to low. The following equation gives the total PWM period [10].

$$\text{PWMperiod} = \frac{1}{\text{clock}} \times 4 \times T2_{-DIV\_BY\_16} \times (\text{period} + 1)$$  \hspace{1cm} (4)

In order to find the on time for a particular speed,

$$\text{Ontime} = \frac{\text{Dutycycle} \times \text{PWMperiod}}{100\%}$$  \hspace{1cm} (5)
The relation between the speed of a motor at no load and duty cycle is

\[ \text{Dutycycle} = \frac{\text{Speed} + 100}{2} \]  

(6)

At 50% duty cycle, the speed of the motor is zero because of PWM and complimentary PWM signals. When duty cycle is over 50%, the speed is positive. On the other hand, when duty cycle is below 50%, the speed is negative. In other words, robot moves forward during the positive speed and backwards during the negative speed. The relation between PWM on time value and actual on time [10] is given by

\[ \text{PWMontime} = \frac{\text{Ontime}}{4} \times \text{clock} \times \frac{1}{T2\_DIV} \]  

(7)

The following list provides the constants that are used for this project.

- \( \text{clock} = 40 \text{ MHz} \)
- \( T2\_DIV = 16 \)
- \( \text{period} = 255 \)

PIC18F4520 was not able to handle these calculations as it requires at least 32 bit registers. Therefore, lookup table was used. Robo 3.1 hardly moves at the speed below 35% of the maximum speed of the motor. Therefore, the lookup table is developed for the speeds from 35% to 100% of the total speed of the motor. The lookup table is given in Table 2.
<table>
<thead>
<tr>
<th>Speed</th>
<th>PWM value</th>
<th>Speed</th>
<th>PWM value</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>173</td>
<td>68</td>
<td>215</td>
</tr>
<tr>
<td>36</td>
<td>174</td>
<td>69</td>
<td>216</td>
</tr>
<tr>
<td>37</td>
<td>175</td>
<td>70</td>
<td>218</td>
</tr>
<tr>
<td>38</td>
<td>177</td>
<td>71</td>
<td>219</td>
</tr>
<tr>
<td>39</td>
<td>178</td>
<td>72</td>
<td>220</td>
</tr>
<tr>
<td>40</td>
<td>179</td>
<td>73</td>
<td>221</td>
</tr>
<tr>
<td>41</td>
<td>180</td>
<td>74</td>
<td>223</td>
</tr>
<tr>
<td>42</td>
<td>182</td>
<td>75</td>
<td>224</td>
</tr>
<tr>
<td>43</td>
<td>183</td>
<td>76</td>
<td>225</td>
</tr>
<tr>
<td>44</td>
<td>184</td>
<td>77</td>
<td>227</td>
</tr>
<tr>
<td>45</td>
<td>186</td>
<td>78</td>
<td>228</td>
</tr>
<tr>
<td>46</td>
<td>187</td>
<td>79</td>
<td>229</td>
</tr>
<tr>
<td>47</td>
<td>188</td>
<td>80</td>
<td>230</td>
</tr>
<tr>
<td>48</td>
<td>189</td>
<td>81</td>
<td>232</td>
</tr>
<tr>
<td>49</td>
<td>191</td>
<td>82</td>
<td>233</td>
</tr>
<tr>
<td>50</td>
<td>192</td>
<td>83</td>
<td>234</td>
</tr>
<tr>
<td>51</td>
<td>193</td>
<td>84</td>
<td>236</td>
</tr>
<tr>
<td>52</td>
<td>195</td>
<td>85</td>
<td>237</td>
</tr>
<tr>
<td>53</td>
<td>196</td>
<td>86</td>
<td>238</td>
</tr>
<tr>
<td>54</td>
<td>197</td>
<td>87</td>
<td>239</td>
</tr>
<tr>
<td>55</td>
<td>198</td>
<td>88</td>
<td>241</td>
</tr>
<tr>
<td>56</td>
<td>200</td>
<td>89</td>
<td>242</td>
</tr>
<tr>
<td>57</td>
<td>201</td>
<td>90</td>
<td>243</td>
</tr>
<tr>
<td>58</td>
<td>202</td>
<td>91</td>
<td>244</td>
</tr>
<tr>
<td>59</td>
<td>204</td>
<td>92</td>
<td>246</td>
</tr>
<tr>
<td>60</td>
<td>205</td>
<td>93</td>
<td>247</td>
</tr>
<tr>
<td>61</td>
<td>206</td>
<td>94</td>
<td>248</td>
</tr>
<tr>
<td>62</td>
<td>207</td>
<td>95</td>
<td>250</td>
</tr>
<tr>
<td>63</td>
<td>209</td>
<td>96</td>
<td>251</td>
</tr>
<tr>
<td>64</td>
<td>210</td>
<td>97</td>
<td>252</td>
</tr>
<tr>
<td>65</td>
<td>211</td>
<td>98</td>
<td>253</td>
</tr>
<tr>
<td>66</td>
<td>212</td>
<td>99</td>
<td>254</td>
</tr>
<tr>
<td>67</td>
<td>214</td>
<td>100</td>
<td>255</td>
</tr>
</tbody>
</table>

Table 2: Lookup Table for Speed and PWM On Time

3.3. PD Controller

The PD controller used for this project has one input and one output. The input is a distance from a wall and the output is a change in a right wheel speed. In order to correct the position of a robot, left wheel speed is kept constant at about 65% of the maximum speed of the motor at no load. Now onwards, speed is indicated in terms of motor speed at no load unless it is specified. On the other hand, the right wheel speed is changed dependent on the position of the robot.
Figure 8 illustrates the control system of Robo 3.1. It has a PD controller which is written in C language using a CCS compiler. Robot represents the plant of a system and feedback is taken from Ultrasonic Distance Meter. Reference is set to 30 cm for the distance from a wall.

![Diagram of the control system of Robo 3.1](image)

The PD controller gets an error as input, which is the difference between the reference value and the actual position of the robot. The output of the PD controller is the right wheel speed based on the error. The robot changes its position in relation to the right wheel speed. For an instance, the right wheel speed decreases if Robo 3.1 is closer to the wall that it follows on its left. On the other hand, the right wheel speed increases if Robo 3.1 is away from the wall. The gains of proportional and derivative controllers are important for a control system. If gains are too high, the system becomes unstable. On the other hand, if gains are too low, the system’s steady state error is large. Matlab simulations in Section 2.3 helped to get the idea of what should be the gains for PD controller.

The entire system can work properly if feedback is trustworthy. The ultrasonic sensors had limitations. The ultrasonic readings were not accurate when the robot was at an angle to a wall. Moreover, taking readings from three ultrasonic sensors took large amount of time and the pulse width of the detected object varied by large amount depending on the distance of an object from the robot. In other words, the shortest time can be as small as 450 µs and largest time can be as big as 60 ms. Therefore, an extra care needed to be taken. There are two ultrasonic sensors that are used on its left. Then, error was computed between those two sensors by taking the difference of their readings. Moreover, an average of those two sensors were taken and compared with the reference value. In addition, a set of different margins were selected. If robot is within 28 and 32 cm away from the wall, then both the motors will run at the nominal (65%) speed. If robot is between 23 and 28 cm, then right wheel speed is decreased. If robot is between 15 and 23 cm, then right wheel speed is decreased with a larger amount. Similarly, if robot is between 32 and 37 cm away from the wall, then the right wheel speed is increased. If the robot is between 37 and 45 cm away from the wall, the right wheel speed is increased by a larger amount. Moreover, if the current error indicates that the robot is moving closer to the reference value compared to the previous error, then the amount of right wheel speed
decreased or increased is small. On the other hand if the current error indicates that the robot is moving away from the reference value as compared to the previous error, then the amount of right wheel speed decreased or increased is large. This explanation holds when front distance is greater than 100 cm. When front distance is less than 100 cm and robot is closer to the wall, the robot turns right for approximately 500 ms at 65% speed. On the other hand, if robot is too away from the wall, it turns left for approximately 250 ms at 65% speed. The ultrasonic sensors gives faulty readings several times, so if robot is within 15 cm from the wall, the robot turns left for approximately 250 ms at 65% speed. Similarly, if robot finds itself more than 45 cm away from the wall, it turns right for approximately 250 ms at 65% speed.

3.4. Integration of Software

After knowing about how each of the components (motors, UDM and PD controller) work, it is essential to integrate all the software and execute them on right time for better performance. It is shown in the flowchart in Appendix B. When robot is switched on, the PIC18F4520 configures CCP1, CCP2 and Timer2 module for PWM signals. Then, robot starts slowly with 50% of the maximum speed of the motors at no load and increases its speed up to 65% of the maximum speed. Robo 3.1 also takes at least one ultrasonic reading to initialize ultrasonic distance variables. Next, PD controller part comes in the picture. Based on the distance variables, PD controller sets the right wheel speed. Then, PIC18F4520 reads the corresponding PWM on time value from the lookup table and sets the PWM on time. Once the PWM on time is set, PWM module controls the speed of the right wheel. The program control starts reading ultrasonic readings and based on these readings speed of the motors are calculated again. The procedure repeats continuously.

4. Conclusion

In this report, a differential drive mobile robot is presented which is controlled with the help of a PD controller to maintain a constant distance from the wall. Matlab simulations were carried out to get the overall idea of the problem. The robot successfully follows a wall. Hence, a PD controller is implemented and tested successfully.

However, the robot could have followed a wall much smoothly if ultrasonic sensors gave accurate readings when robot was at an angle with the wall. Moreover, a few sensors can be mounted on the right side of the robot, so that robot can travel through narrow paths like in the corridor. A further research can be done where better sensors can be used. In addition, a 32 bit microcontroller or DSPIC can be used, so that calculations for on time of a PWM signal can be performed by the robot itself rather than looking up in a table. Also, a microcontroller that has more CCP modules can be used, so that ultrasonic sensors can be controlled through those modules along with the motors.
References


Appendices

A. Software Listing

1. Integrated code

```c
#include <18f4520.h>
device ICD=TRUE
#fuses HS,NOLVP,NOWDT
#use delay (clock=40000000)
#use rs232 (stream=LCD, baud=9600, xmit=PIN_B5, rcv=PIN_B4)
#include <ctype.h>
#include <stdlib.h>
define delay_s(x) delay_ms(x*1000) // delay in seconds
#define velocity_cm 34442.4 // velocity of ultrasonic wave in cm/s
#define velocity_in 13560 // velocity of ultrasonic wave in inches/s
#define instruction_time (40000000/4) // clock divide by 4
#define speed_50 192 // PWM on time for 50% speed
#define speed_65 211 // PWM on time for 65% speed
#define speed_75 224 // PWM on time for 75% speed
#define PWM_period 0.0004096 // Total PWM time period

// Global Variables
char display_lcd[15]; // characters displayed on LCD
int32 left_distance, front_distance, right_distance; // from
   // ultrasonic sensors
   // left_distance is left front sensor reading
   // right_distance is left back sensor reading
int16 overflow_count; // overflow count for timer 1
int value = 0; // PWM on time value in terms of #
   // between 0 to 255
signed int Speed = 0; // Desired speed
int leftturn, rightturn = 0; // number of left and right turns

// Subroutines
void lcd_putchar(char c[], int d); // display characters on LCD
void convert_display_lcd(int32 x, int y); // convert integers into
   // ascii characters
void get_value(); // get PWM on time value for speed
void ultrasonic(); // read distances
void move_fwd(); // move forward
void move(); // move forward in the beginning
void increase (int amount); // increase speed by amount in %
void decrease (int amount); // decrease speed by amount in %
void turn_right(int time); // turn right for x amount of time (x is
   // called time)
void turn_left(); // turn left

// Interrupts
#define_timer1 // interrupt for timer overflow
void timer1_isr(); // interrupt service routine
```
{ 
    overflow_count++; // increment overflow counter 
}

void main()
{
    // local variables
    signed int32 error = 0; // error between two left sensors
    signed int32 prev_error = 0; // keep record of previous error
    signed int32 error_diff = 0;
    int32 average = 0; // average of two left sensors

    float x_ref = 30; // reference value
    output_low(PIN_D2); // disable motors
    delay_s(5); // 5 second delay

    setup_ccp1(CCP_PWM); // Configure CCP1 as a PWM
    setup_ccp2(CCP_PWM); // Configure CCP2 as a PWM
    setup_timer_2(T2_DIV_BY_16, 255, 16); // Configure time period for
    // PWM

    Speed = 50; // initiate the robot with 50% speed
    get_value(); // get PWM on time value for 50% speed
    move(); // robot moves forward with 50% speed
    ultrasonic(); // read distances

    Speed = 60; // increase speed to 60%
    get_value();
    move(); // robot moves forward with 60% speed

    // controller code
    while(True)
    {
        prev_error = error; // update previous error
        error = (left_distance - right_distance); // difference between
        // two left sensors
        average = (left_distance + right_distance)/2; // average of two
        // left sensors

        // obstruction in front and too close to the wall and at an angle
        if (error <= -2 && average <= 23 && front_distance < 100)
        {
            turn_right(500); // turn right for 500 ms
        }
        // robot is moving away from reference value and is closer to the wall
        // and at an angle
        else if (error <= -2 && error < prev_error && average <= 23 &&
                 average > 15)
        {
            decrease(15);
        }
        // robot is at the same position and is closer to the wall and at an
        // angle
else if (error <= -2 && error == prev_error && average <= 23 && average > 15)
{
    decrease(0);
}

// robot is moving closer to the reference value and is closer to the wall and at an angle
else if (error <= -2 && error > prev_error && average <= 23 && average > 15)
{
    decrease(5);
}

// robot is moving away from reference value and is closer to the wall
else if (error < prev_error && average < 28 && average > 23)
{
    decrease(10);
}

// robot is moving closer to reference value and is closer to the wall
else if (error > prev_error && average < 28 && average > 23)
{
    decrease(5);
}

// robot is at the same position and is closer to the wall
else if (error == prev_error && average < 28 && average > 23)
{
    decrease(0);
}

// obstruction in front and too away from the wall and at an angle
else if (error >= 2 && error > prev_error && average >= 37 && front_distance < 100)
{
    turn_left();
}

// robot is moving away from the reference value and is away from the wall and at an angle
else if (error >= 2 && error > prev_error && average > 37 && average < 45)
{
    increase(10);
}

// robot is at the same position and is away from the wall and at an angle
else if (error >= 2 && error == prev_error && average > 37 && average < 45)
{
    increase(0);
}

// robot is moving closer to the reference value and is away from the wall and is at an angle
else if (error >= 2 && error < prev_error && average > 37 && average < 45)
{
    increase(5);
}

// robot is moving away from the reference value and is away from the wall
else if (error > prev_error && average > 32 && average < 37)
// robot is moving closer to the reference value and is away from the
// wall
else if (error < prev_error && average > 32 && average < 37)
{
    increase(5);
}

// robot is at the same position and is away from the wall
else if (error == prev_error && average > 32 && average < 37)
{
    increase(0);
}

// robot is at an angle less than 90 degrees with respect to x axis and
// closer to the wall
else if (error >= 2 && error > prev_error && average < 15)
{
    increase(1);
}

// robot is at an angle greater than 90 degrees with respect to x axis
// and closer to the wall
else if (error >= 2 && error < prev_error && average < 15)
{
    decrease(1);
}

// robot is at an angle greater than 90 degrees with respect to x axis
// and away from the wall
else if (error >= 2 && error > prev_error && average > 45)
{
    decrease(10);
}

// if robot is too close to the wall decrease right wheel speed by the
// largest amount,
// However, it's been checked again after this sequence and corrected by
// a turn.
else if (error >= 2 && error < prev_error && average < 15)
{
    decrease(20);
}

// for minor corrections in angle
else if (error <= -2)
{
    decrease(1);
}

// for minor corrections in angle
else if (error >= 2)
{
    increase(1);
}

// else nominal speed when robot is between 28 and 32 cm away from the
// wall
else
{
    Speed = 65;
    get_value();
    move_fwd(); // set right wheel speed to 65%
// robot is too close to the wall (for immediate correction)
if (average < 20 )
{
  turn_right(250);
}

// obstruction detected in front and robot is between 28 and 32 cm
else if (front_distance <= 50 )
{
  turn_right(500);
}

// robot is too away from the wall (for immediate correction)
else if (average > 40 )
{
  turn_left();
}

// else nominal speed when robot is between 28 and 32 cm away from the
// wall
else
{
  Speed = 65;
  get_value();
  move_fwd(); // increase right wheel speed to 75%
}

void turn_right(int time)
{
  if (rightturn <= 2)
  {
    set_pwm1_duty(speed_65); // This sets the on time for PIN #
      // 17(left motor) nominal speed
    set_pwm2_duty(45); // This sets the on time for PIN #
      // 16(right motor)
    output_high(PIN_D2); // enable motors
    delay_ms(time);
    output_low(PIN_D2); // disable motors
    ultrasonic(); // get UDM reading
    rightturn++; // increment right turn count
  }
  else
  {
    rightturn = 0; // reset the right turn count
    Speed = 65; // set nominal speed
    get_value();
    move_fwd(); // move forward
  }
}

void turn_left()
{
  if(leftturn <= 2)
  {
    set_pwm1_duty(45); // This sets the on time for PIN #
      // 17(left motor) nominal speed
  }
set_pwm2_duty(speed_65); // This sets the on time for PIN #16 (right motor)
output_high(PIN_D2); // enable motors
delay_ms(250);
output_low(PIN_D2); // disable motors
ultrasonic(); // get UDM reading
leftturn++; // increment left turn count
}
else
{
    leftturn = 0; // reset left turn count
    Speed = 65; // set nominal speed
    get_value();
    move_fwd(); // move forward
}
}
void decrease(int amount)
{
    Speed = Speed - amount; // decrease speed by the amount
    get_value();
    move_fwd(); // decrease right wheel speed
}

void increase (int amount)
{
    Speed = Speed + amount; // increase speed by the amount
    get_value();
    move_fwd(); // increase right wheel speed
}

void get_value() // lookup table for PWM on time
{
    int32 temp_value = 0;

    if (Speed == 0)
        value = 128;
    else if (Speed < 35)
        value = 160;
    else if (Speed == 35)
        value = 173;
    else if (Speed == 36)
        value = 174;
    else if (Speed == 37)
        value = 175;
    else if (Speed == 38)
        value = 177;
    else if (Speed == 39)
        value = 178;
    else if (Speed == 40)
        value = 179;
    else if (Speed == 41)
        value = 180;
    else if (Speed == 42)
        value = 182;
    else if (Speed == 43)
        value = 183;
else if (Speed == 44)
    value = 184;
else if (Speed == 45)
    value = 186;
else if (Speed == 46)
    value = 187;
else if (Speed == 47)
    value = 188;
else if (Speed == 48)
    value = 189;
else if (Speed == 49)
    value = 191;
else if (Speed == 50)
    value = 192;
else if (Speed == 51)
    value = 193;
else if (Speed == 52)
    value = 195;
else if (Speed == 53)
    value = 196;
else if (Speed == 54)
    value = 197;
else if (Speed == 55)
    value = 198;
else if (Speed == 56)
    value = 200;
else if (Speed == 57)
    value = 201;
else if (Speed == 58)
    value = 202;
else if (Speed == 59)
    value = 204;
else if (Speed == 60)
    value = 205;
else if (Speed == 61)
    value = 206;
else if (Speed == 62)
    value = 207;
else if (Speed == 63)
    value = 209;
else if (Speed == 64)
    value = 210;
else if (Speed == 65)
    value = 211;
else if (Speed == 66)
    value = 212;
else if (Speed == 67)
    value = 214;
else if (Speed == 68)
    value = 215;
else if (Speed == 69)
    value = 216;
else if (Speed == 70)
    value = 218;
else if (Speed == 71)
    value = 219;
else if (Speed == 72)
value = 220;
else if (Speed == 73)
    value = 221;
else if (Speed == 74)
    value = 223;
else if (Speed == 75)
    value = 224;
else if (Speed == 76)
    value = 225;
else if (Speed == 77)
    value = 227;
else if (Speed == 78)
    value = 228;
else if (Speed == 79)
    value = 229;
else if (Speed == 80)
    value = 230;
else if (Speed == 81)
    value = 232;
else if (Speed == 82)
    value = 233;
else if (Speed == 83)
    value = 234;
else if (Speed == 84)
    value = 236;
else if (Speed == 85)
    value = 237;
else if (Speed == 86)
    value = 238;
else if (Speed == 87)
    value = 239;
else if (Speed == 88)
    value = 241;
else if (Speed == 89)
    value = 242;
else if (Speed == 90)
    value = 243;
else if (Speed == 91)
    value = 244;
else if (Speed == 92)
    value = 246;
else if (Speed == 93)
    value = 247;
else if (Speed == 94)
    value = 248;
else if (Speed == 95)
    value = 250;
else if (Speed == 96)
    value = 251;
else if (Speed == 97)
    value = 252;
else if (Speed == 98)
    value = 253;
else if (Speed == 99)
    value = 254;
else if (Speed >= 100)
    value = 255;
void move_fwd()
{
    set_pwm1_duty(speed_65);  // This sets the on time for PIN # 17(left
        // motor)nominal speed
    set_pwm2_duty(value);     // This sets the on time for PIN #
        //16(right motor)
    output_high(PIN_D2);      // enable motors
    ultrasonic();             // get UDM reading
}

void move()
{
    set_pwm1_duty(value); // This sets the time for PIN # 17(left motor)
    set_pwm2_duty(value); // This sets the time for PIN # 16(right motor)
    output_high(PIN_D2); // enable motors
}

// LCD was used for only debugging purpose
void lcd_putstr(char c[], int d) // function for LCD display
{
    output_high(PIN_B5);       // set pin high to be a serial port
    delay_ms(100);             // pause for serial LCD to initialize
    fprintf(LCD, "%c", 0x11); // turn the back light on
    delay_ms(500);
    fprintf(LCD, "%c", 0x0C); // clear the LCD screen
    delay_ms(500);

    switch(d)
    {
        case 1: fprintf(LCD, "Left-side Distance: %s cm", c);
                delay_ms(500);
                break;
        case 2: fprintf(LCD, "Front Distance: %s cm", c);
                delay_ms(500);
                break;
        case 3: fprintf(LCD, "Right-side Distance: %s cm", c);
                delay_ms(500);
                break;
        case 7: fprintf(LCD, "value: %s cm", c);
                delay_ms(500);
                break;
        case 8: fprintf(LCD, "%s", c);
                delay_ms(500);
                break;
        default: fprintf(LCD, "Invalid Command");
                delay_ms(500);
    }
}
void convert_display_lcd(int32 x, int y)
{
    itoa(x,10,display_lcd);       // integer to ascii string
    lcd_putchar(display_lcd,y);   // display character string on LCD
}

void ultrasonic()          // function for ultrasonic sensors
{
    int32 time;             // travel time for sonar
    int32 distance;        // distance between ultrasonic sensor and obstacle
    int x;                // counter for different ultrasonic sensor
    int32 pin = PIN_D4;   // different pin for each ultrasonic sensor
    int32 ultra[3];       // right, center, left distance

    setup_timer_1(T1_INTERNAL);      // using timer1
    enable_interrupts(int_timer1);   // enable specific timer interrupt
    for(x = 0; x < 3; x++)      // three ultrasonic sensors fired one
        // after another
    {
        enable_interruption();  // enabling global interrupt controls
            // whether interrupts are serviced

        /* input pulse of 10us is required
to turn-on ultrasonic range finder */
        output_low(pin);  // pin kept low for 1 ms
        delay_ms(1);
        output_high(pin); // pin kept high for 10us for pulse
        delay_us(10);
        output_low(pin);  // pin brought back to low
        // again to finish the pulse

        /* Since same pin is used to transmit and receive a signal,
PIN_C7 needs to be checked continuously for high */
        while(!input(pin));     // wait for signal to go high
        set_timer1(0);          // set timer1 to zero and turn it on
        // as soon as signal goes high
        overflow_count = 0;     // initialize overflow_count identifier

        while(input(pin));     // wait for signal to go low
        disable_interruption(); // disable global interrupts

        time = get_timer1();    // get the pulse width

        /* add 65535 when overflow_count is 1 to the original time */
        time = time + ((int32)overflow_count<<16);

        time -= 15;              // subtract overhead
        time /= 2;               // remove return trip

        distance = velocity_cm * time; // distance between ultrasonic
            // sensor and obstacle

        /* clock tick for timer1 is 10MHz.
        divide by 10MHz to get time in seconds. */
division in the end gives less error, since
more significant numbers can give closer value */
distance /= instruction_time; // get original distance in cm
ultra[x] = distance;
pin++;
}

left_distance = ultra[0];       // left_front distance
front_distance = ultra[1];      // front distance
right_distance = ultra[2];      // left_back distance

2. Code for Motor and PWM signal (Test Version) [10]
(Note: Notice the clock speed.)

#include <18F4520.h>
#define ICD = TRUE
#define fuses HS,NOWDT,NOPROTECT,NOLVP
#define use delay(clock=3686400)
#define rs232 (DEBUGGER)

void main()
{
    float desired = 50;
    float calculated = 0;
    int value = 0;

    setup_ccp1(CCP_PWM);   // Configure CCP1 as a PWM
    setup_ccp2(CCP_PWM);   // Configure CCP2 as a PWM

    //   The cycle time will be (1/clock)*4*t2div*(period+1)
    //   In this program clock=3686400 and period=63 (below)
    //   selections the cycle time is:
    //     (1/3.6M)*4*1*64 = 69.4 us or 14.4 khz
    setup_timer_2(T2_DIV_BY_1, 127, 1);

    calculated = (desired+100)/2;
    value = 1.28 * calculated;

    set_pwm1_duty(value);     // This sets the time the pulse is
    // high each cycle.
    // the high time will be:
    // if value is INT:
    //    value*4*(1/clock)*t2div
    // for example a value of 30 and t2div
    // of 1 the high time is 12us

    set_pwm2_duty(value);

} //main
3. Code for Ultrasonic (Test version) [10]  
(Note: Notice the clock speed.)

```c
#include <18F4520.h>             /* pic 18f4520 */
#define clock 20000000
#include <ctype.h>
#define velocity_cm 34424.4   /* velocity of ultrasonic wave in cm/s */
#define velocity_in 13560  /*velocity of ultrasonic wave in inches/s */
#define instruction_time clock/4

int16 overflow_count;
#define clock 20000000
#define velocity_cm 34424.4   /* velocity of ultrasonic wave in cm/s */
#define velocity_in 13560  /*velocity of ultrasonic wave in inches/s */
#define instruction_time clock/4

/*    hex code for lighting LEDs in seven-segment
    display corresponding to the numbers            */
byte CONST LED_MAP[10] =
{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x67};

int32 overflow_count;
#define clock 20000000
#define velocity_cm 34424.4   /* velocity of ultrasonic wave in cm/s */
#define velocity_in 13560  /*velocity of ultrasonic wave in inches/s */
#define instruction_time clock/4

void timer1_isr()             /* interrupt service routine */
{
    overflow_count++;          /* increment overflow counter */
}

void display_number(int32 n)  /* display for distance measurement on
    3-digit seven-segment LED display */
{
    int32 i = 0;

    output_b(LED_MAP[n/100]);  /* leftmost (first) seven-segment display */
    output_low(PIN_C0);     /* turn-on leftmost seven-segment
    LED display and keep others off */
    delay_ms(2);
    output_high(PIN_C0);   /* turn-off leftmost LED display */

    i = n%100;
    output_b(LED_MAP[i/10]);  /* centeral (second) seven-segment display */
    output_low(PIN_C1);     /* turn-on centeral seven-segment
    LED display and keep others off */
    delay_ms(2);
    output_high(PIN_C1);   /* turn-off centeral LED display */

    output_b(LED_MAP[i%10]);  /* rightmost (third) seven-segment display */
    output_low(PIN_C2);     /* turn-on rightmost seven-segment
    LED display and keep others off */
    delay_ms(2);
    output_high(PIN_C2);   /* turn-off rightmost LED display */
```

Page 28 of 34
void main() {
    int32 time;
    int32 distance;

    setup_timer_1(T1_INTERNAL);  /* using timer1 */
    enable_interruptions(int_timer1); /* enable specific timer interrupt */

    delay_ms(5000); /* 5 sec delay in the beginning to give enough time to hardware to get ready */

    do
    {
        int32 c;
        enable_interruptions(global); /* enabling global interrupt controls whether interrupts are serviced */

        /* input pulse of 10us is required to turn-on ultrasonic range finder */
        output_low(PIN_C3); /* PIN_C3 kept low for 1 ms */
        delay_ms(1);
        output_high(PIN_C3); /* PIN_C3 kept high for 10us for pulse */
        delay_us(10);
        output_low(PIN_C3); /* PIN_C3 brought back to low again to finish the pulse */

        /* Since same pin is used to transmit and receive a signal, PIN_C3 needs to be checked continuously for high */
        while(!input(PIN_C3)); /* wait for signal to go high */
        set_timer1(0); /* set timer1 to zero and turn it on as soon as signal goes high */
        overflow_count = 0; /* initialize overflow_count identifier */

        while(input(PIN_C3)); /* wait for signal to go low */
        disable_interruptions(global); /* disable global interrupts */

        time = get_timer1(); /* get the pulse width */

        /* add 65535 when overflow_count is 1 to the original time */
        time = time + ((int32)overflow_count<<16);

        time -= 15; /* subtract overhead */
        time /= 2; /* remove return trip */

        distance = velocity_cm * time; /* distance between ultrasonic sensor and obstacle */

        /* clock tick for timer1 is 5MHz. divide by 5MHz to get time in seconds. 
        division in the end gives less error, since more significant numbers can give closer value */
        distance /= instruction_time;
    } while(1);
}
for(c = 0; c <= 200; c++)
    display_number(distance); /* display number on 7-segment
        display */
}

while (TRUE);

4. Matlab Simulation Code

% Initial Conditions
theta = pi/2     % angle between robots vertical axis and x axis
Vr0 = 9.14      % Right wheel speed
x = 15 ;        % Distance between wall and the robot

% Constants
K = 0.5;       % proportional constant
D = 150;         % Derivative constant
I = 0;      % Integral constant
L = 18.5;        % distance between two wheels
Vl = Vr0      % Left wheel speed
del_t = .02;    % Time interval between 2 UDM readings
x_ref = 20;    % Reference distance of 20 cm
final_t = 20;    % final time
array_size = ceil(final_t/10*del_t) % array size for x position
     and time

%Initialized Variables
x_array = zeros(1,array_size)  % array for x values
t_array = zeros(1,array_size)  % array for time values
e = 0;     % error
i = 1;     % count for array indices
integral_control = 0;   % integral controller gain

for t = 0: del_t: final_t  % for loop

% Formula for change in x position
prev_e = e;    % error(t-delta_t)
e = x_ref - x;
del_e = e - prev_e;   % delta_e = error - error(t-
     delta_t)
prev_integral_control = integral_control;

% sum of old integral control
integral_control = prev_integral_control + I*del_e* del_t;

%Controller
del_Vr = -K*e - D*del_e - integral_control;    % delta_Vr
\[ V_r = V_{r0} + \Delta V_r; \]

\[ x_{\text{dot}} = \frac{(V_l+V_r)/2}{(\pi/2 - \theta)}; \]
\[ \theta_{\text{dot}} = \frac{(V_r-V_l)}{L}; \]

\[ x = x + x_{\text{dot}} \Delta t; \]
\[ \theta = \theta + \theta_{\text{dot}} \Delta t; \]

\[ x_{\text{array}}(i) = x; \quad \% \text{array of x values to plot} \]
\[ t_{\text{array}}(i) = t; \quad \% \text{array of time values to plot} \]

\[ \text{plot} \]

\[ i = i+1; \]

\[ \text{end} \]

\[ \text{plot}(t_{\text{array}},x_{\text{array}}) \]
\[ \text{grid} \]
B. Flowchart of Software

Start

Configure CCP1, CCP2 and Timer2

Slowly start the robot and increase its speed to nominal speed gradually and read distance once to initialize the distance variables

while (1)

Controller

Stop
C. Parts List

<table>
<thead>
<tr>
<th>Part</th>
<th>Manufacturer</th>
<th>Quantity</th>
<th>Cost $US/item</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC18F4520</td>
<td>Microchip</td>
<td>1</td>
<td>10.00</td>
</tr>
<tr>
<td>Motors</td>
<td>Solarbotics</td>
<td>2</td>
<td>7.00</td>
</tr>
<tr>
<td>Plastic Chassis Material</td>
<td>Various</td>
<td>1</td>
<td>10.00</td>
</tr>
<tr>
<td>Ultrasonic Sensor</td>
<td>Parallax</td>
<td>3</td>
<td>30.00</td>
</tr>
<tr>
<td>Battery Packs (AA)</td>
<td>Eagle Plastic Devices</td>
<td>2</td>
<td>2.00</td>
</tr>
<tr>
<td>LCD</td>
<td>Parallax</td>
<td>1</td>
<td>30.00</td>
</tr>
<tr>
<td>Batteries</td>
<td>Various</td>
<td>16</td>
<td>2.00</td>
</tr>
<tr>
<td>PCB manufacture</td>
<td>ExpressPCB</td>
<td>1</td>
<td>50.00</td>
</tr>
<tr>
<td>Misc. electronics and connectors</td>
<td>Various</td>
<td>1</td>
<td>20.00</td>
</tr>
</tbody>
</table>

D. Circuit Schematic