MXEN 2002

Project Report

Kee-An Seet
19776219@student.curtin.edu.au
Harry Cassidy
20607591@student.curtin.edu.au

Contents

List of Figures

List of Tables

1 Nomenclature

2 The Task

We were tasked with building, wiring, and programming a robot to complete a series of objectives, such as navigating a maze autonomously and moving a camera with a servo as to let the operator identify targets. We were to use the Arduino Mega hardware with a DC motor drive system along with several distance sensors to help achieve the objectives.

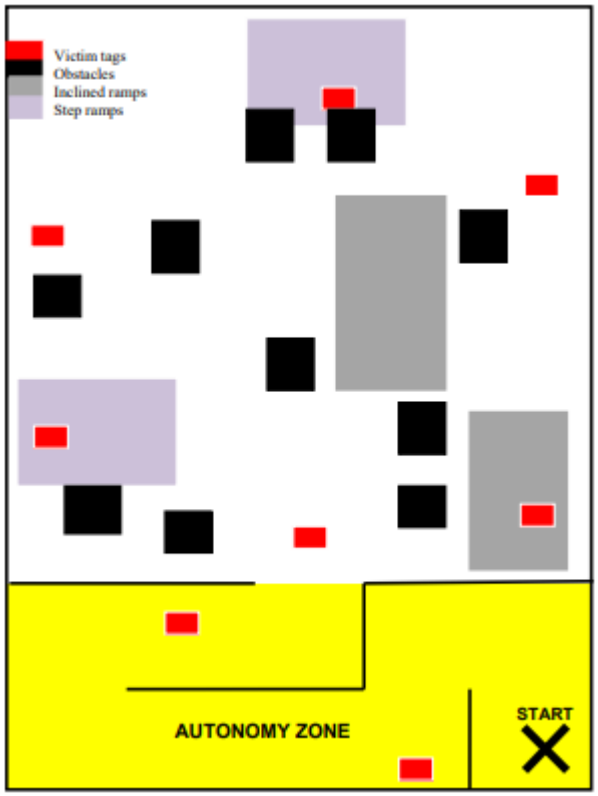

Figure 1: Supplied example of the task map

2.1 Autonomy Zone

The "Autonomy Zone" is the section of the map in which the robot is completely self controlled. The idea behind this challenge is that in the event of a connection dsiruption in a dangerous area for humans, the robot should be able to rescue it's self from danger without external control.

2.2 Manual Control

The majority of the task is to control the robot through wireless communication. During this period the robot (controlled by the user) must drive through the zone and send a camera feed back to the user for "victim" tags to be recorded.

3 Our Solution

3.1 Overview

Mechanically constructed from the supplied robot base. It has two driving motors each attached to tank treads, on top there is a platform with mounting holes for an Arduino Mega and various sensor brackets. Initially our plan was to use the given prototyping breadboard with limited modification to the drivebase, however after some testing we identified several issues with this. The final product we created used a more permanent Arduino "Shield" that we created with a solder on prototyping board.

3.2 Issues and Solutions

Before starting we identified several potential issues and solutions that we thought may impact our project, listed below:

Issue	Solution
During Lab G we found that the tracks would fall off if run too long as the nut would unscrew and as such unbolt the wheel	Initially we intended on using lock-tite, however we didn't want to use a potentiallly damaging product, to solve this issue we opted to use a second nut on each bolt to reduce the effect of vibration
We found throughout our labs we found that we had many issues with the wiring and using the bread-boards as the connections were loose	To remedy this we planned out our circuit and soldered a "prototyping shield" for an Arduino Mega to cre-ate our own "MXEN2002 shield"
The front sensor didnt have the res-olution to sense the in front of the robot from its initial position	We moved the sensor mounting around as to position it below the robot and sunk back slightly

Table 1: Issues + Solutions

3.3 Joystick Drive

To be able to control the robot manually, we implemented a joystick drive system, where joystick inputs on the controller would result in movement of the robot. Since the raw joystick outputs have a range from 0-1024, they were first scaled by a factor of 1/4 (to bring it within 0-256 range) and then transmitted to the robot via the XBees. These values were processed on

the robot's end by mapping them from 0 - 256 to -378 - 378, which were used to calculate 2 variables, leftValue and rightValue, where $leftValue = joyY + joyX$ and $rightValue = joyY - joyX$. This achieves a differential drive effect, where both left and right motors react to the joystick Y position for forwards and backwards drive, but each react in opposite magntidues depending on the joystick X position, resulting in the turning of the robot.

To convert these drive values into values that can run the motors, we opted for a PWM drive system, where PWM signals are used to control the speed of each DC motor independently.

3.4 PWM Calculations

We aimed to have our duty cycle range between 0.05s and 0.1s. A Phase Correct PWM mode was chosen as it allowed for the widest range of possible values and prescaler values. We also settled on prescaler value of 8 as it allowed the resulting COMP value to be within the 0-10000 range that the ADC can read at.

Phase Correct PWM: f(base) = f(microcontroller)/(2*TOP*PRE)
TOP = 10000, PRE = 8
$f(base) = (16 * 10^6)/(2 * 20000 * 8) = 50Hz$
$Period = 0.02s$
$DutyCyclerange : 0.05s - 0.1s$
$DutyCycle = COMP/TOP$
$COMPrange : (20000 * 0.05) - (20000 * 0.01) = 1000 - 2000$

With this information we can choose which bits to set the PWM registers to. Our drivetrain used the TCCR1A and TCCR1B registers.

As we decided to use PWM Phase correct (mode 10), we set the WGM11 and WGM13 bits in the PWM registers to 1. This mode meant that our TOP value of 20000 would be set in the ICR1 register. We want the PWM to clear on up count and compare on down count, so we set the COM1B1 bit to 1. Our prescaler is 8 which corresponds to setting only the CS11 bit to 1.

The following table shows the bits set in the TCCR1A and TCCR1B PWM registers that control the drivetrain:

Register	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	W
TCCR1A	1	0	1	0	0	0	1	0

Table 2: TCCR1A register

Register	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10
TCCR1A	0	0	0	1	0	0	1	0

Table 3: TCCR1B register

3.5 Abandoned features

Throughout the construction of our robot we developed several ideas that we didnt end up using for various reasons. ...

4 Program logic flowcharts

4.1 Robot.c program logic

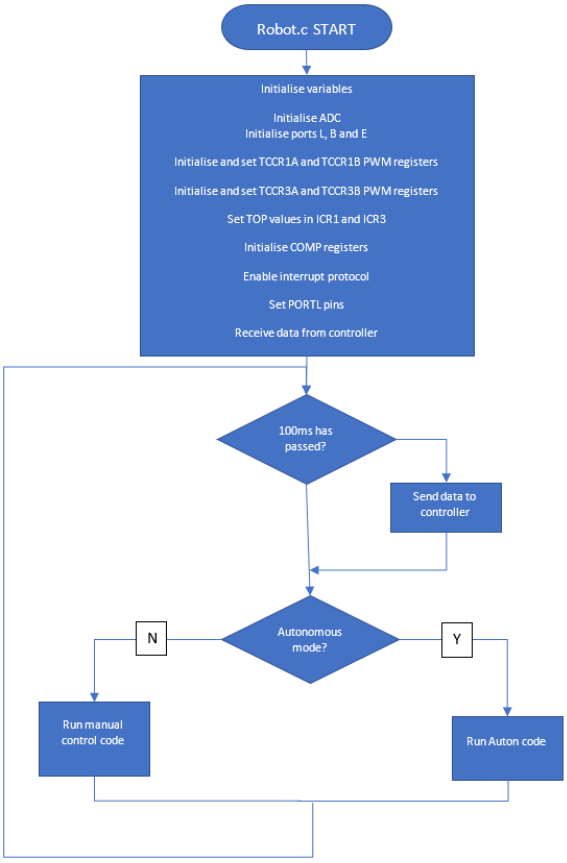

Figure 2: Flowchart of Robot.c code

4.2 Autonomous code program logic

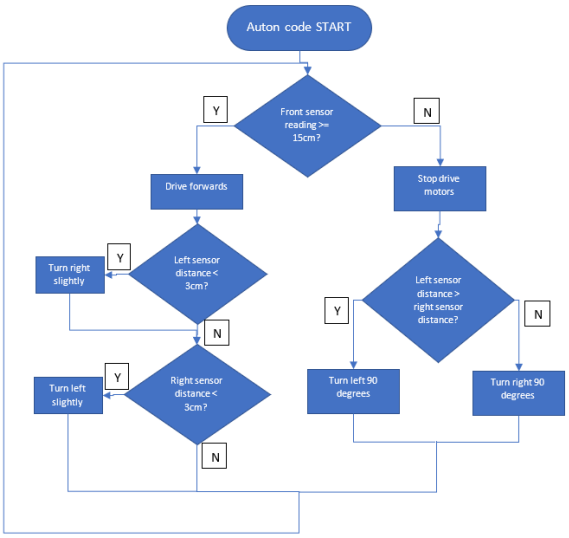

Figure 3: Flowchart of autonomous mode code