Software porting for Bearcat III robot

A thesis submitted to the Division of

Graduate Studies and Advanced Research

Of the University of Cincinnati

in partial fulfillment of the

requirements for the degree of

MASTER OF SCIENCE

In the Department of Mechanical, Industrial and Nuclear Engineering

of the College of Engineering 2004

by

Rema Vinjamuri

B.Tech., Mechanical Production Engineering

University of Kerala

SCT College of Engineering, Trivandrum, India, 2001

Thesis Advisor and Committee Chair: Dr. Ernest L. Hall

ABSTRACT

The purpose of modern technology is to make the technology easier and accessible to every person in the world. The Bearcat Robot is an autonomous ground vehicle which is operated by a number of programs which are in MS Dos mode. These programs can be understood only by a person who knows the software language. The main purpose of this project is to make these programs run on Windows 2000/XP, the most commonly used operating system. The programs which are in Turbo C language are ported to Turbo C++ and later upgraded to Microsoft Visual C++, so that they are compatible with Windows 98/2000/XP systems. Later, these programs are tested to confirm whether they work in a Windows 98/2000/XP environment. This change is necessary for the interaction of the programs to any new interface with which even a layman could operate the robot.

ACKNOWLEDGEMENTS

First and foremost I would like to acknowledge the help provided by my advisor Dr. Ernest L. Hall. Without his guidance and support this thesis would not have been possible. His suggestions, feedback and constant encouragement have been the greatest help in completion of this thesis.

I would like to thank Dr. Richard Shell and Dr. Ronald Huston for agreeing to serve on the thesis committee. I would also like to thank them for their suggestions and positive feedback.

Last but surely not the least I would like to thank each member of the Robotics Team, both present and past for their help and support they have given me during the period of my graduate study at the University of Cincinnati. Your contribution, support and suggestions have been the single largest factor towards the improvement of my work.

Table of Contents

List of Figures

Chapter 2

1. Introduction

1.1 Background

With global outsourcing there is a pressing need for manufacturers of US to come up with quality products in limited time. There is a lot of demand for products which can be manufactured in bulks, with less inventory and ‘first time right’ concept applied. Due to this critical need for increased productivity and high quality there is the introduction of robotic automation. Manufacturers initially used special machines for the production. But due to the inflexibility and high production costs of these machines there is a high interest in the use of robots which are capable of performing a variety of manufacturing tasks in more flexible working environment, at greater profits and lower costs.

Robot Institute of America defines a robot as “A reprogrammable, multifunctional manipulator designed to move material, parts, tools, or specialized devices through various programmed motions for the performance of a variety of tasks”.

Through the time, robots have advanced and have become part of many industrial companies. Making the robots self-directed is challenging and of great interest to many sectors of industry from modern machine shops, medical field, construction, manufacturing, space exploration to homes.

The autonomous robots are a technical challenge. It is also of great technical interest because it raises challenging and rich computational issues. These new concepts of broad usefulness are emerging. Lately the development of autonomous vehicles has found increasing applications in military, making of smart machines and highway transportation.

The development of these technologies necessary for the robots which are autonomous is a very huge challenge and very difficult to automate the logic, the control and the sensitivity of the system.

The Center for Robotics Research at University of Cincinnati has been a regular participant in the annual International Ground Vehicles Competition (IGVC) organized by the Association for Unmanned Vehicle Systems International (AUVSI). The University of Cincinnati's robot, called the Bearcat III has won a lot of prizes in the competition.

Over the years, Bearcat III has provided an excellent test bed for technologies being investigated and developed at the Center for Robotics Research. One of the major issues involving the development of an autonomous vehicle is successful obstacle avoidance. For the "Autonomous Navigation" event in the IGVC, a simple algorithm has been developed to avoid specific configuration of obstacles when the Bearcat navigates between two white lines which form a track [1].

1.2 Objective

The object of this thesis is to change the existing software programs of the Bearcat III, which are in a MS Dos environment, to a Window 98/2000/Xp environment, which would be more accessible and understable to any person who would want to understand and operate the robot.

1.3 Organization of Thesis

Chapter 2 studies the different components of the autonomous Bearcat III robot. It explains the functions of various robotic parts and how they integrate together to form a fully functional unit. In chapter 3, the software functions which guide the robotic movements are explained. This chapter also elaborates the changes necessary to port the software code from Turbo C to Microsoft VC++ environment. Chapter 4 talks about the future direction and the scope of the thesis work.

2. The Bearcat III

2.1 Introduction

The Intelligent Ground Vehicle Competition (IGVC) is one of three, unmanned systems, student competitions that were founded by the Association for Unmanned Vehicle Systems International in the 1990s. The IGVC challenges engineering student teams to integrate advanced control theory, machine vision, vehicular electronics, and mobile platform fundamentals to design and build an unmanned system for competition against both U.S. and international teams. IGVC teams focus on developing a suite of dual-use technologies to equip ground vehicles of the future with intelligent driving capabilities.

2.2 System components of Bearcat III

The Bearcat mobile robot is a sophisticated, intelligent, controllable, programmable system. The adaptability of the robot primarily depends on the conceptual, analytical and architectural design of the sensing and controlling system used. The main components of the Bearcat III robot [2, 5] are (1) electrical system, (2) vision system, (3) mechanical system, (4) GPS system, (5) object tracking system, and (6) steering/motion control system. Apart from these there are also safety and health monitoring systems. All these systems are connected to and controlled by a Pentium based computer running Microsoft Disk Operating System (DOS).

A block diagram of the system

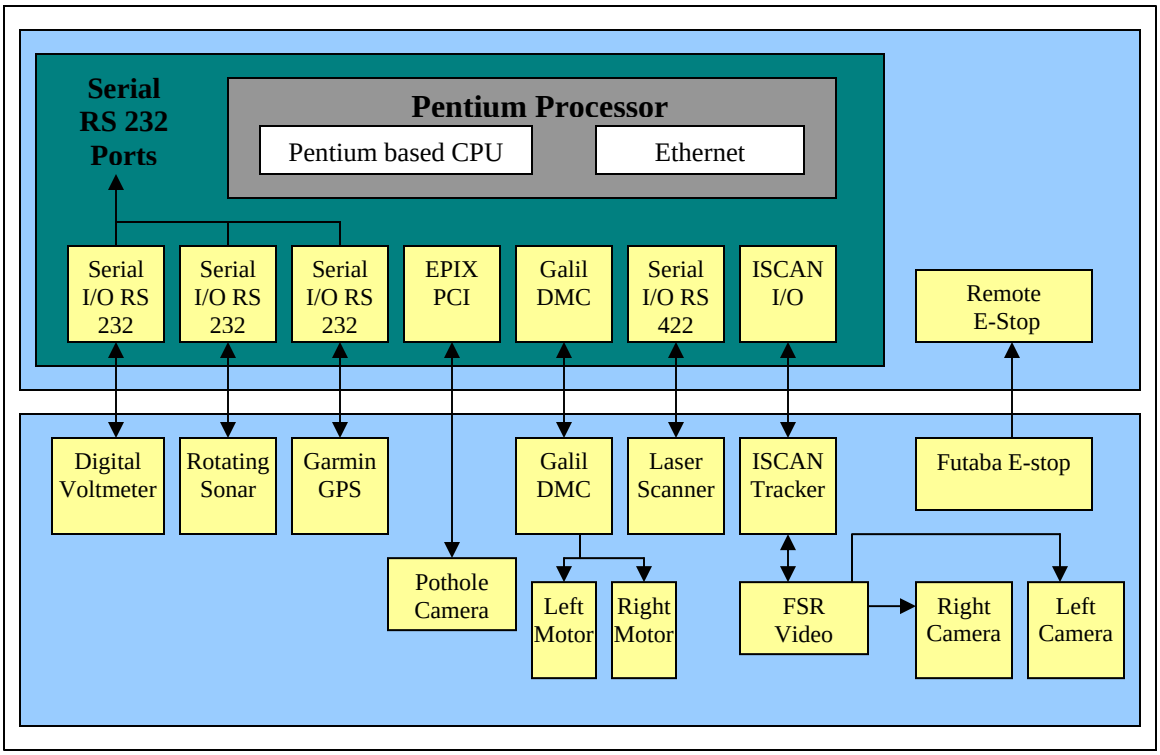

Figure 2.2: Bearcat III – Block Diagram

2.2.1 Structure

The robot base is constructed from an 80/20 Aluminum Industrial Erector Set. Bearcat III is steered using two independent 36 Volt, 12 Amp motors. Each of these motors drives one of the wheels independently using a gearbox. The gearbox in the transmission system helps to amplify the motor torque by about 20 times. The power for the individual motor is supplied by a BDC 12 amplifier, which actually amplifies the signal from the Galil DMC motion controller. A position encoder mounted on each of the drive motors completes the control loop. A castor wheel supports the rear of the robot, which is free to swing when the robot has to negotiate a curve. The control of the motion is done by the usage of differential speed drive wheel.

2.2.2 Electrical System

The electrical system of Bearcat III consists of a DC power system and an AC power system. Both these systems derive power from three 12-volt DC 130 amp-hrs deep cycle batteries connected in series. A 36-volt DC 600-watt power inverter provides 60 Hz pure sine wave output at 115 volts. The inverter supplies AC electrical power for all AC systems including the main computer, cameras, and auxiliary regulated DC power supplies. The heart of the power system being the solenoid acts as a switch, which can be controlled to cut off the power during emergency.

2.2.3 Vision System

The vision system defines the components that assist in the line following function of Bearcat III along with pothole detection. It consists of two JVC digital cameras mounted on either side of the robot, one pothole detection camera mounted at the top, a video switching unit to switch between the camera visions. Image processing is done by the ISCAN tracking device. Image co-ordinates obtained by the ISCAN are two-dimensional while actual world co-ordinates are three-dimensional. The mapping of the two dimensional points and reorganization of the three dimensional image is done by the vision calibration.

2.2.4 Mechanical System

The mechanical system as a whole serves as steering control for the robot. Bearcat III is an outdoor vehicle designed to carry a payload of 100 pounds. The components include 40:1 reduction gearbox, two pairs of flexible couplings, two 36 volts servomotors and two sets of wheels with shafts, couplings and keys. The computer through Galil motion

controller controls the servomotors, which supply power to the gear train for the mechanical motion transmission. Two separate gearboxes are used to individually power the wheels. The self-locking mechanism of the worm gears does not require the vehicle to have a separate mechanical breaking system. Power is transmitted to the front wheels. The rear castor wheel gives the Bearcat a zero turning radius.

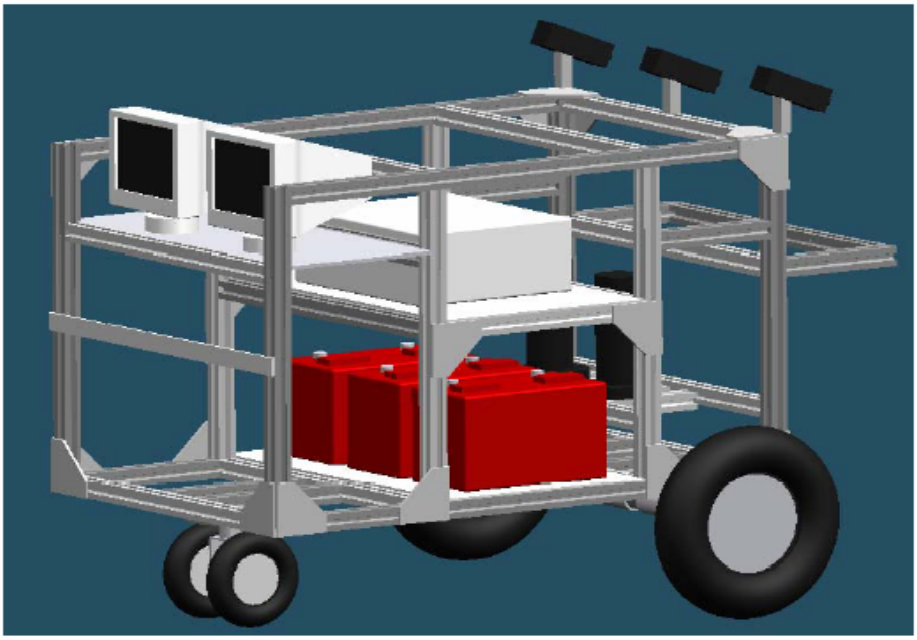

Figure 2.2.4: The Mechanical System

2.2.5 GPS System

Global navigation is the ability to determine one's position in absolute or referenced map co-ordinates and to move to a desired destination point. The Bearcat uses a Motorola GPS to navigate from one point to the other. The GPS tracks the NAVSTAR constellation of satellites. The satellite signals received by an active antenna are tracked with 12 parallel channels of L1. C/A code is then down converted to an IF frequency and

digitally processed to obtain a full navigation solution of position, velocity, time and heading. The solution is then sent over the serial link via the 10-pin connector.

2.2.6 Object Tracking System

The Bearcat has two different systems for object detection. One is the rotating sonar sensor and the other is the laser scanner system.

- Sonar System

Apart from the vision system for line tracking the sonar system is used for object detection. It is powered by a 12 Volts DC, 0.5 Amps power unit. The two main components of the ultrasonic ranging system are the transducers and the drive electronics. The sonar sensor is mounted on a brushless DC servo motor for rotation in different directions.

- Laser Scanner System

The Laser scanner LMS-200 is the new enhancement in the object tracking system on the. LMS 200 is a non-contact measurement system that scans its surroundings two dimensionally. LMS works by measuring the time of flight of laser light pulses. The shape of the object is determined by the sequence of impulses received.

Both these systems are described in detail in further chapters.

2.2.7 Motion Control System

Two Electro-craft brush-type DC servomotors drive the two wheels independently. The encoders provide the feedback from the system. The two drive motors are operated in parallel using the Galil MSA 12-80 amplifiers. The Galil DMC 1030 motion control board is the main controller card of the system and it is controlled through a computer.

The motion control of the Bearcat robot has the ability of turning about its drive axis, which is called Zero Turning Radius (ZTR). This feature offers extensive maneuverability and can negotiate very sharp turns with much ease. The ZTR functional capability is achieved by turning wheels in opposite directions. When one is rotated forward and the other is rotated backward the robot could turn around its own drive axis. By rotating the wheels at differential speeds BEARCAT III is able to negotiate curves smoothly.

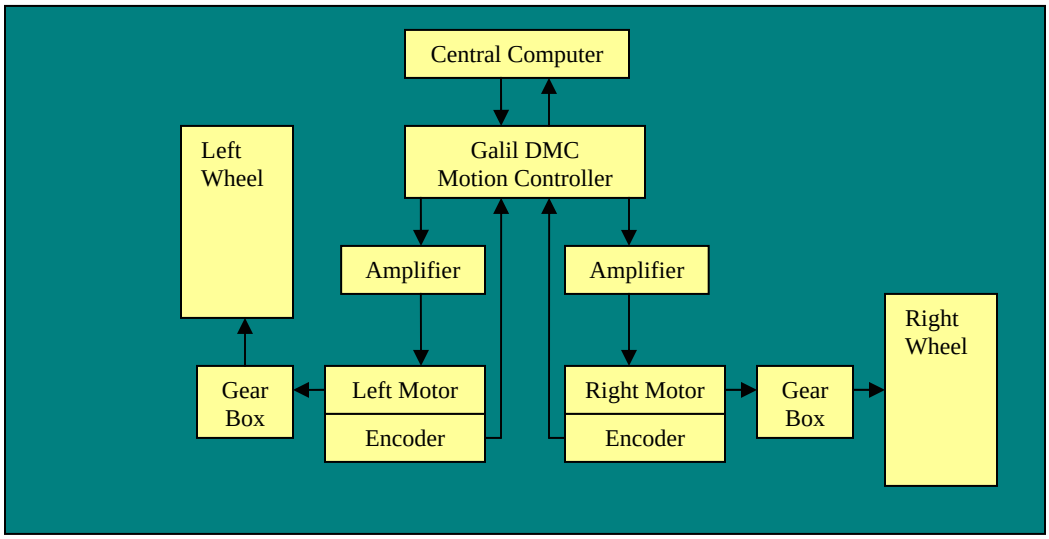

Figure 2.2.7: Motion Control System - Block Diagram

2.2.8 Safety System

- Manual Emergency Stop

The manual emergency stop unit consists of a red manual push button located on the easily accessible rear surface of the vehicle. When pressed, the power to the motors is cut off and the self-locking mechanism of the gearbox brings the vehicle to an instant halt. The self-locking mechanism ensures that the vehicle does not move when it is not powered, and serves as a safety measure against any undesirable motion such as rolling when parked on a slope.

- Remote Controlled Emergency Stop

The mobile robot must be de-activated by a remote unit from a distance of no less than 50 feet in compliance with the rules for this contest. The remote controlled emergency stop consists of a Futaba transmitter, a receiver, an amplifier and a relay. The advantage of using this is that the transmitter need not be in line with the sight of the receiver. The Futaba transmitter uses a 6V DC and transmits FM signals at 72.470 MHz over a range of 65 feet. This amplified current activates the contacts of the relay that in turn activates the emergency stop solenoid and cuts power to the motors.

- Health Monitoring System

The Bearcat III is equipped with a self-health monitoring system. A RS 232 serial port is used to take input from a digital multi meter, which can be accessed from C++ code to check the total DC voltage of the batteries. The health monitoring is implemented as a C++ class module that has methods that can monitor battery voltage and display warning messages to the computer screen. Two voltage threshold trip points can be set that will

trigger a low voltage and a critical low voltage warning message. The low voltage warning indicates that the battery voltage is below the first threshold trip point and that preparations should be made to change or charge the batteries. The critical low voltage warning indicates that corrective actions must be taken immediately because robot power system shutdown is eminent. The voltmeter class can also be used in code to sound an audible alarm or activate the robot strobe light at the specified threshold point. The power system voltage display is also visible to the operator and provides a constant indication of the robot electrical voltage.

2.2.9 Overall System Integration

For the autonomous challenge-line following the inputs comes from the vision system: as image coordinates of the track to be followed, Obstacle avoidance system: as laser scanner/sonar data and pothole detection data. These inputs are processed by the central controller to give commands to the motion control system, which drives the mechanical drive train. For the Navigational challenge data from the navigation system: as the GPS data and from the obstacle avoidance system: as the laser scanner/sonar data are used as inputs by the central controller to give commands to the motion control system which drives the mechanical drive train.

3. Changing the Program Execution Environment

3.1 Introduction

Currently the software programs governing the movement and operation of the Bearcat Robot are implemented in C in Turbo C environment. This severely limits the features and advantages offered by other high level languages and also affect the execution of the programs. Using C instead of C++ devoid the programs of benefits of object oriented programming. The inherent advantages of using C++ for robotics include [1]:

Availability: C++ can be used on a variety of computers ranging from workstations, personal computers to modern age supercomputers.

Portability: The standardization of C++ by ANSI committee has made it a universal language followed by the software vendors.

Speed: Some people contest the idea of using C over C++ because of the execution speed, since C is more close to system level programming. However with the speed of modern computers C++ offers equivalent speed as C.

Graphical Applications: Modern engineering now demands the simplicity to execute software applications using graphics in robotics. The C++ language supports is not only suited for formulas and computations, it can be used effectively in data base and business applications.

Reusability: Developing programs in C++ increases the reusability of the code thus saving time, reducing software maintenance and making it more robust.

Object Orientation: To solve problems involving large and complex programs, the Object Oriented Programming (OOP) methodology simplifies the programming design and implementation effort.

The above points elucidate the advantages of using C++ over C; however without the support of a sophisticated compiler we still cannot make the best utilization of the C++ features. By using 32 bit Microsoft VC++ compiler over 16 bit Turbo C compiler, we achieve the following benefits:

Visual Libraries: By using Microsoft VC++ compiler we can link our code to languages providing advanced support to Visual Libraries like VB.Net, Java etc. This linking of programs offers a user friendly interface which can be used by a lay man to execute the underlying programs. Visual interface hide the internal code implementation from the user and increase the usability of the program. Robotic applications are now widely used in commercial industries. In that kind of environment it is essential that the robotic movements are controlled by a Graphic User Interface and not by command line arguments. To facilitate this operation the support of Visual Libraries is extremely important.

Visual debugger: Robotic Applications comprise of many lines of code. For such a huge application program, it is important that we use sophisticated compilers for debugging purposes. Since in Microsoft VC++ we have the aid of Visual Debugging, it aids in locating and fixing bugs much faster.

32 bit vs 16 bit: Recently 32 bit programming has become more prevalent than before than 16 bit compilers. With the arrival of Windows, more and more of the new programs for Windows will be 32-bits instead of 16-bits. By using a 32 bit compiler the data

execution speed of the programs will be greatly enhanced, thereby reducing the latency between instruction fetching and instruction execution. This feature is extremely important to increase the response time of the robot for time critical operations. Also in case of thread implementation a 32 bit compiler gives us much stronger support than a 16 bit compiler.

CODE FLOW DIAGRAM

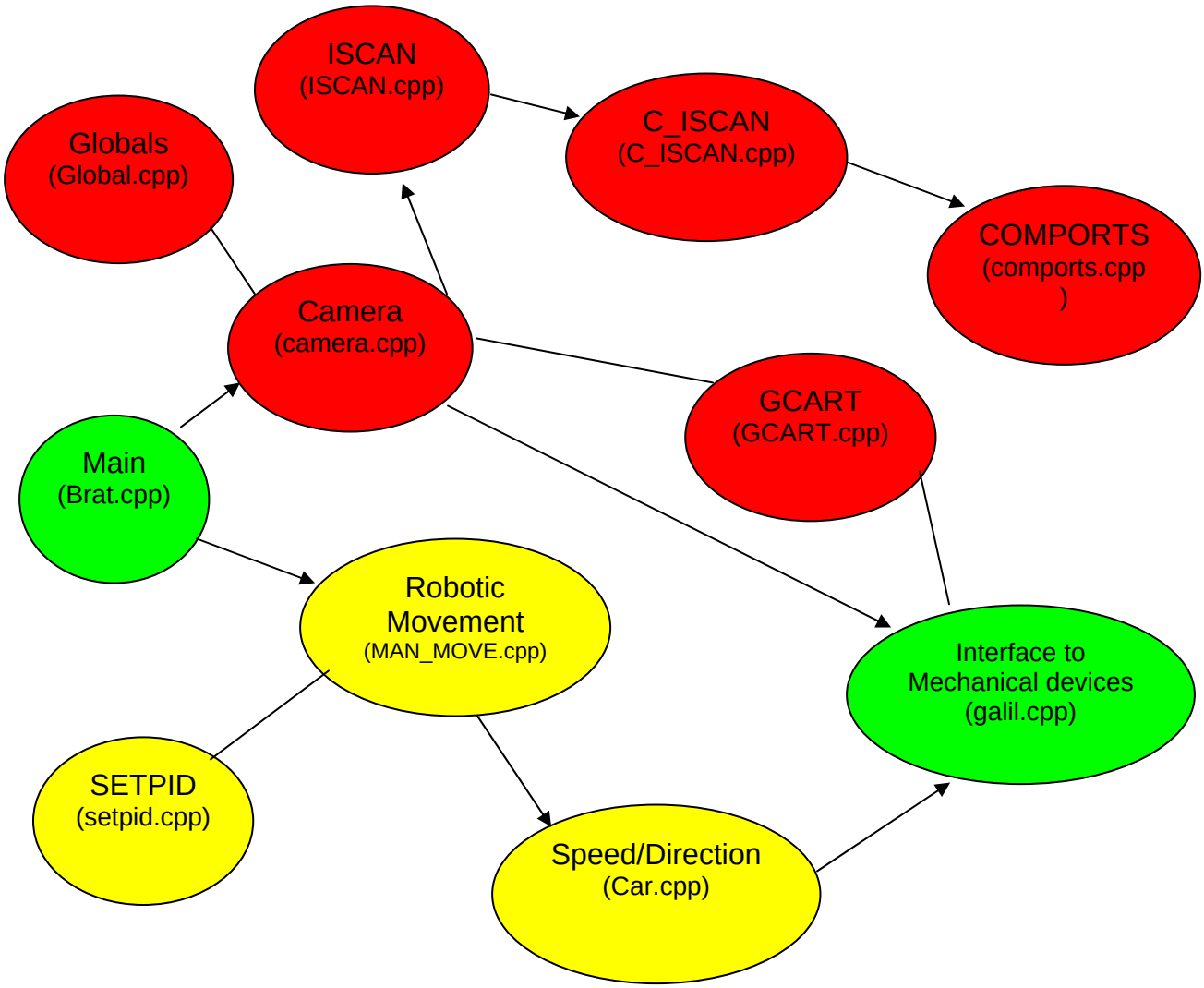

Figure 3.1: code flow diagram

3.2 VARIOUS COMPONENTS

3.2.1. Main (Brat.cpp)

The Main execution of the program begins from this point. This is the entry point of the robotic application which passes control to other program components for execution. The basic body of the program comprises of a switch statement which prompts the user to perform certain actions. The actions namely are as follows;

Robotic Movement in manual mode

Processing data provided by camera

Control on entry and exit of the application

3.2.2. Manual Movement

This particular file handles the movement of the robot. Whenever the control passes to this function, it prompts the user of the various parameters to guide the robotic movement. It commands the robot by issuing the following commands:

Increase speed incrementally

Decrease speed incrementally

Soft Stop

Hard Stop

Turn Right

Turn Left

3.2.3. Robotic speed control (car.cpp)

When the Manual movement function (man_move.cpp) is invoked by the main application, it prompts the user about the actions to be taken to steer the robot. The manual movement function in turn passes control to car.cpp to perform the necessary actions. It has an interface with galil.cpp (explained below) which controls the mechanical movement of the robot. The function car.cpp provides the following features:

speedCARx(long int spx): This command when invoked will set the speed of bearcat III in the x coordinate direction. It is called with a value between 0 - 250000.

speedCARy(long int spy): This command when invoked will set the speed of bearcat III in the y coordinate direction. It is called with a value between 0 - 250000.

stopCAR(): This command is used to stop the movement of the robot.

steerCAR(float): This function when invoked will put the steering wheel to the absolute angle given. This angle ranges between +-20.

auto_steerCAR(float): This function when invoked will put the steering wheel to the absolute angle given. This angle ranges between +-20.

startCAR():This function when called will initialize the galil board.

3.2.4. Functions to communicate with Galil DMC (galil.cpp)

This function provides a software interface to control the robotic motor movement. Depending on the commands passed it controls the mechanical motion of Bearcat III. It provides the following interface handles to the user:

writeDMC(char a): This function when invoked will pass a character to DMC.

ResetDMC(): This function resets the DMC, thus clearing the stored buffer variables so that ths operation does not interfere with the new commands.

readDMC(): This function is used to read the DMC_base integer value.

writeStatus(char a): This function is used to write the DMC_status.

readStatus(): This function is used to read the integer value of the DMC_status.

galilHasInfo(): This function returns the hash information by sending the fifth bit of the DMC_status value.

galilIsReady(): Performs to check if the DMC is ready to take the information.

sendDMC(char str1[]): This function when invoked will pass a string to DMC.

download (char str1 []): This function will download a file from the PC to the DMC buffer.

getDMC(): This function when invoked will confirm data is available and the will retrieve it.

fromDMC(): This function when invoked will check to see if there is a character available and if so will return it.

getDMCValue();

inDMC(char str1[40]): This function when invoked will send a request for data, once the command is recieved it will wait on the data, and display it.

submitDMC(char str1[15]): This function when invoked will send the DMC a command, and if the command is legal and there are no errors it will say so.

clearPC (): This function when called clears the PC buffer.

clearDMC (): This function when called clears the DMC buffer.

initDMC (): This function when invoked will prepare the DMC by initializing it to 128 FIFO.

set_upDMC (): This function when called sends the necessary settings for proper operation of the steering and drive motors.

one_rotation (): This Function when invoked will inform one full rotation of the wheel.

inDMCreturn (char str1 [40]): This function when invoked will send a request for data, once the command is received it will wait on the data, and return the value as a double.

3.2.5. Camera operation

This file provides handles to execute the camera operations with respect to an obstacle. The various member functions it utilizes to achieve and their operations are described below:

- VisionTransform (int Camera,int Test): This function is used to calculate the real world coordinates corresponding to the image coordinates of the object.

- MakeMoveAI (int Camera): This function positions the camera in to be on track with the obstacle object.

- SetValues (int Camera): This function is used to set the camera attributes.

- InitStuff (): This function is used for initializing the camera variables.

- AngleOK (): This algorithm will check all angles of lines calculated from the ISCAN line following routine found in this file at the Method Vision Transform. It will maintain an array of the average angle of the line from the robot. It will compare the current line

angle to the average line angle in the array. If the difference is too large then the line measure is considered to be in error. A not OK is returned as int IsAngleOK. If the angle is not OK then the array should be reset and the camera switched. The array of line angles should be very small and consideration should be given as to how much an angle should be different from the base line in order to be considered an erroneous reading.

- ResetAngleOK (): This function is used to reset the angle system as set by the previous method.

This file consists of a global class with data and member functions. These member functions provide useful utilities to perform operations common to all other components. For example writing log data to a log file, logging the current status, storing the current robotic position etc.

3.2.6. Scanning data (ISCAN.cpp)

This file provides functionalities to scan the obstacle image. It has the following member functions to perform the operations:

- auto_mode_test (): This function runs the test in the auto mode.

- iscan_calibration (void): This function is used to calibrate the menu function. It calibrates the left arrow, right arrow, up arrow, down arrow and the escape character.

- test_iscan (): This function tests the scan calibration.

- init_iscan (): This function initializes the scan parameters.

3.2.7. Serial port interface to ISCAN (C_ISCAN.cpp)

This is the Iscan Tracker class which is inherited from the COMPORT class. This class interfaces with the Iscan Video Tracker. This interface is used to send commands and read data from the Iscan tracker unit via the serial port. Various commands can be sent to change the Iscan Tracker settings. XY position data is read from the Iscan unit. The Iscan factory set Baud rate is 38400.

- readData (outputdata *): Data Output Request - This function sends the Data Output Request command to the Iscan and Reads data from serial port then fills the output data DATA structure with the data packet information

- newDataAvailable (void): Data Output Request - This function sends the Data Output Request command to the Iscan and Reads data from serial port then fills the output data DATA structure with the data packet information [0 = No new Data, 1 = Yes, New data Available].

- sendcommand (int command[]): Sends two byte commands to Iscan.

- change_y_pos (int ypos1): To alternate gate window y-position function

- printData (void): Prints contents of scanData array to the screen.

3.2.8. Communication ports (COMPORTS.cpp)

Implementation of COMPORT class uses the interrupt method to receive data from the COM ports (serial ports). Communication with COM1 and COM2 are supported but only under dos compilation.

- SetPortx (int Port): Set Port Number base address. Valid values Table COM1 = 0X3F8 or COM2 = 0X2F8.

- SetBaud (int Baud, int BuadHi): Set Port Baud Rate. Valid values Speeds Table (115 200 BPS 57600 BPS 38400 BPS 19200 BPS 9600 BPS 4800 BPS 2400 BPS 600 BPS 300 BPS 50 BPS).

- SetParameters (int Wordlen, int StopBit, int Parity): Function to set Line control Register and to set communication parameters. Parameters Table Word Length =5, 6, 7, or 8 bits, Stop Bit = 1 Stop Bit or 2 Stop bits, Parity Select: No_Parity = 0, Odd_Parity= 1, Even_Parity= 3, High_Parity= 5, Low_Parity = 7.

- SetByteSize(int ByteSize): Set Byte Size of serial port data. The byte size values are 1 byte, 4 bytes, 8 bytes, 14 bytes.

- void SetMCR (int DTR, int RTS, int OUT1, int OUT2): Function to Set Modem control register. The modem control register table has the following data: (DTR = ON or OFF (Data Terminal Ready), RTS = ON or OFF (Request to Send), OUT1 = ON or OFF (Aux Output 1), OUT2 = ON or OFF (Aux Output 2), Set OUT2 to ON to Enable interrupts).

- WritetoPortint (int t[], int size): This function sends integers to serial port.

- WritetoPort (char t[], int size): This function sends character(s) to serial port.

- OpenComPort (void): This function is used to Set com port register settings.

- readComPort (void): Reads data from comport buffer array.

- readComPort (void): Reads one character of data from comport buffer array.

- readComPort2 (char * outch): Reads characters of data from comport

- ReadFile (char buffer [], int NumberofBytes): Reads the specified number of bytes from the serial port and stores the data in the buffer array

- CloseComPort (void): Restore comport settings.

- getData (void): Get current value of character just read from serial port (ch) data member.

- printParameters (void): Display to screen a listing of the current com port settings and parameters.

3.2.9. Data logging (GLOBALS.cpp)

This file is used to open the output streams for logging the data.

3.2.10. Initializing Global variables (G-CART.cpp)

This file is used to initialize the global variables.

3.2.11. Setting the PID values (Setpid.cpp)

The function is used to set the PID constants for the motor.

3.3. PORTING

3.3.1 Delay Functions

Delay functions are used to introduce time delays in the program execution. This is a system call which makes the program execution to halt for the time as passed in the function argument. During this time the thread which executes the delay function just sits idle, and other threads get the time share for CPU execution. In Turbo C, this delay is introduced using a delay () system library call. In Microsoft VC++ environment, we use the analogous sleep () system library call to delay the execution of the program.

Since sleep () is a part of the Win32 API we can use it in any compiler targeting the windows platform. The argument passed in sleep function determines the number of milli-seconds to delay running the program. It suspends a task object unconditionally (that is, it puts the task object in the IDLE state). When the object is no longer pending, the task is rescheduled. In MSVC++ we have to include the header file “windows.h” to call a sleep () function.

3.3.2. Clearing the Screen

To clear the screen buffer we need to use library functions. In turbo C environment it is achieved by calling the clrscr () function. For this function to work we have to include the header file “conio.h” in the main program. The analogous to using clrscr () in MSVC++ is system (“CLS”). This basically prompts the system to clear the display screen buffer. We need to include the header file dos.h to execute the system library routine.

3.3.3. Moving to a X and Y location

The function gotoxy() in turbo C environment shifts the cursor over to the given coordinates within the virtual screen. The gotoXY procedure takes two arguments, the Column and Row and then moves the cursor to the given coordinates within the virtual screen. Microsoft VC ++ does not provide us with a direct call to execute the gotoxy function. To achieve the similar result we need to include the header file windows.h and write a function as shown below:

```
void gotoxy( int x, int y)
{
COORD Coor;
HANDLE hOutput ;
hOutput=GetStdHandle(STD_OUTPUT_HANDLE);
Coor.X =x ;
Coor.Y = y;
SetConsoleCursorPosition(hOutput,Coor);
}
```

COORD: The COORD structure defines the coordinates of a character cell in a console screen buffer. The origin of the coordinate system (0, 0) is at the top, left cell of the buffer. The structure definition of COORD is as mentioned below. It has two data members, one for the x-coordinate and the other for the y-coordinate.

```
typedef struct _COORD
{
        short X;
short Y;
} COORD,
*PCOORD;
```

GetStdHandle (): The **GetStdHandle** function retrieves a handle for the standard input, standard output, or standard error device. It has the following declaration:

```
HANDLE GetStdHandle (
        DWORD nStdHandle
);
```

If the function succeeds, the return value is a handle to the specified device, or a redirected handle set by a previous call to **SetStdHandle**. The handle has GENERIC_READ and GENERIC_WRITE access rights, unless the application has used **SetStdHandle** to set a standard handle with lesser access.

If the function fails, the return value is INVALID_HANDLE_VALUE. To get extended error information, call **GetLastError**.

If an application does not have associated standard handles, such as a service running on an interactive desktop, and has not redirected them, the return value is NULL.

Handles returned by **GetStdHandle** can be used by applications that need to read from or write to the console. When a console is created, the standard input handle is a handle to

the console's input buffer, and the standard output and standard error handles are handles of the console's active screen buffer. These handles can be used by the **ReadFile** and **WriteFile** functions, or by any of the console functions that access the console input buffer or a screen buffer (for example, the **ReadConsoleInput**, **WriteConsole**, or **GetConsoleScreenBufferInfo** functions). [2]

SetConsoleCursorPosition (): The **SetConsoleCursorPosition** function sets the cursor position in the specified console screen buffer. It has the following structure:

```
BOOL SetConsoleCursorPosition (
    HANDLE hConsoleOutput;
    COORD dwCursorPosition;
);
```

If the function succeeds, the return value is nonzero.

If the function fails, the return value is zero. To get extended error information, call **GetLastError**.

The cursor position determines where characters written by the **WriteFile** or **WriteConsole** function, or echoed by the **ReadFile** or **ReadConsole** function, are displayed. To determine the current position of the cursor, use the **GetConsoleScreenBufferInfo** function. If the new cursor position is not within the boundaries of the console screen buffer's window, the window origin changes to make the cursor visible. [3].

4. Future Direction

By changing the execution environment from Turbo C to Microsoft VC++, there could be a link in software code to graphic user interfaces written in VB.Net, Java etc. This is possible because MSVC++ is highly compatible with the previously mentioned languages. By operating the Robot with a GUI, the internal implementation of the code can be hidden to a lay man, thus increasing the usability of the code for various applications.

5. References

[1] Sachin Modi, ‘Comparison of three obstacle avoidance methods for an autonomous guided vehicle’, Master of Science Thesis

[2] http://www.aspire.cs.uah.edu/textbook/introcomp6003.html

[3]http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dllproc/base/getstdhandle.asp

[4]http://msdn.microsoft.com/library/default.asp?url=/library/enus/dllproc/base/setconsolecursorposition.asp

[5] Pravin Chandak, “Study and Implementation of Follow the Leader”, Master of Science Thesis

[6] Mayank Saxena, Obstacle Avoidance using Laser Scanner for Bearcat III, Master of Science Thesis