CS 684 Project Report

Group #5

Greenhouse Monitoring And Harvesting Using Accurate And Automated Bot Guidance System

Devendra Bhave (114050004)

Mohd Vasimuddin (114050007)

Meenakshi Verma (123050014)

Mukund Lahoti (123050018)

November 18, 2012

Contents

List of Figures

List of Tables

1 Introduction

Greenhouse is the building in which plants are grown. Building is covered with various covering materials like plastic sheets or glass. This allows solar or artificial light to enter the building, but traps the heat inside the building. Weather conditions inside a greenhouse can be controlled. Our project aims to harvest crops in such greenhouses completely autonomously. This report discusses design and implementation aspects of the project. It talks about design principles applied, engineering choices made and risks mitigated. It also enumerates difficulties faced during the project development and further possible enhancements.

1.1 Definitions, Acronyms and Abbreviations

- FireBird: A robot indigenously designed at ERTS laboratory, IIT Bombay. [1]
- AVR-libc: Standard C library implementation by AVR Systems
- ABGS: Automatic Bot Guidance System
- PWM: Pulse Width Modulation

2 Problem Statement

Project consists of two key entities – greenhouse and user. A single greenhouse consists of greenhouse building with plants arranged in aisles and troughs formation, greenhouse controller and the Bot. There could be multiple such greenhouses each one equipped with its own greenhouse controller and the Bot. The entity *user* refers to human farmer. Greenhouse controller autonomously manages greenhouse functions like sunlight control using sun shades, humidity control using blowers, etc. It has its own set of sensors, actuators and control logic. User merely specifies its parameters like amount of sunlight needed for plants, permissible humidity range, etc. Operation of such greenhouse controller is out of scope for this project.

Each greenhouse has one Bot to carry out certain farming tasks. Bot consists of FireBird robot with mounted wireless network camera and robotic arm. Figure 1 shows design of the Bot. User communicates with Bot via suitable technology. Such communication method

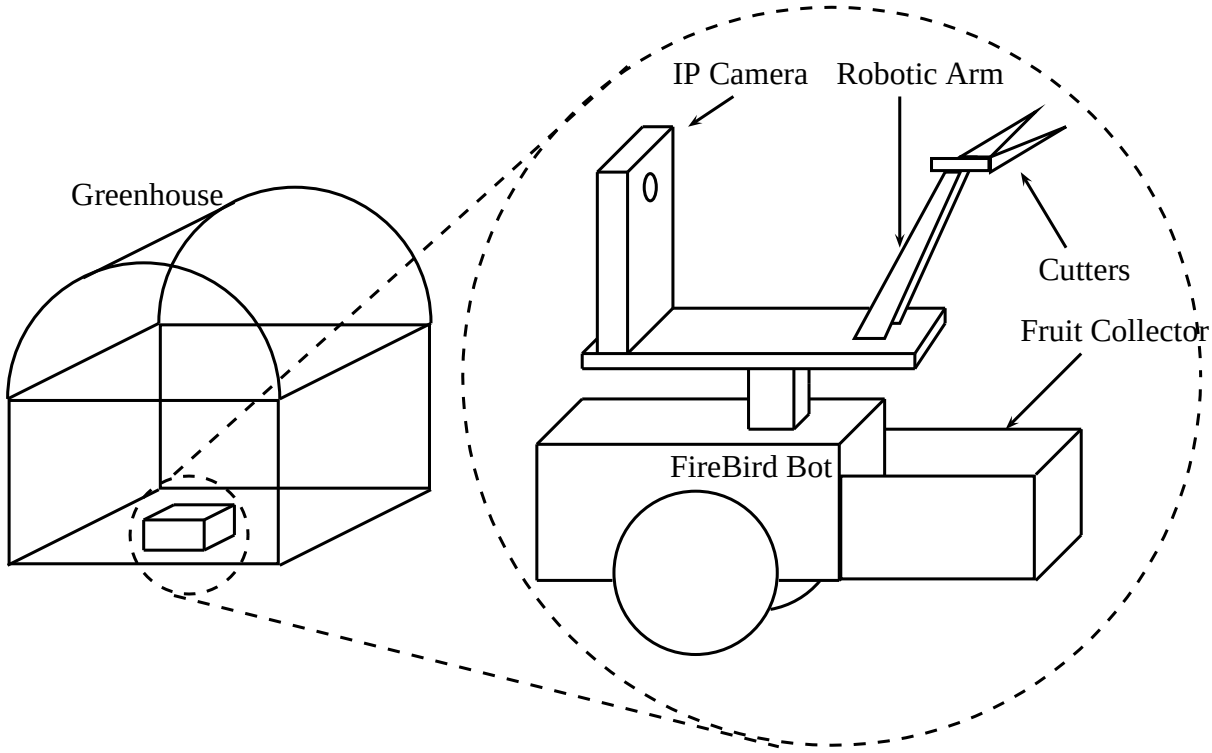


Figure 1: Design of FireBird Bot

is assumed to be present. We have used ZigBee point-to-point link for communication between Linux machine and FireBird bot. User assigns work to the Bot. Bot has capability to move anywhere inside the greenhouse, to take photos and videos of plants. It may communicate with greenhouse controller, monitor plant growth, harvest crops and alert user when necessary.

3 Requirements

3.1 Hardware Requirements

- FireBird V robot
- Linux machine
- Mounted wireless network camera with WiFi

- WiFi access point with Internet connectivity
- Robotic arm with cutter
- Two ZigBee cards

3.2 Functional Requirements

1. User remotely connects to desired greenhouse.
2. User specifies trough number and automatic Bot guidance system maneuvers the Bot to reach given trough.
3. User receives live streaming video from the Bot.
4. Bot uses cutter to cut fruits and vegetables from branches which drop into collector below.
5. User gets battery level indication and energy estimates for each of the above task.

3.3 Non Functional Requirements

1. Intuitive GUI design
2. Good quality video streaming
3. High speed Internet connection

3.4 Design Constraints

User cannot control more than one Bot at a time. There is no support for command broadcasting to multiple Bots. Area of greenhouse is determined by ZigBee range.

4 Implementation

Given the degree of complexity involved, we have divided problem statement into three subtasks:

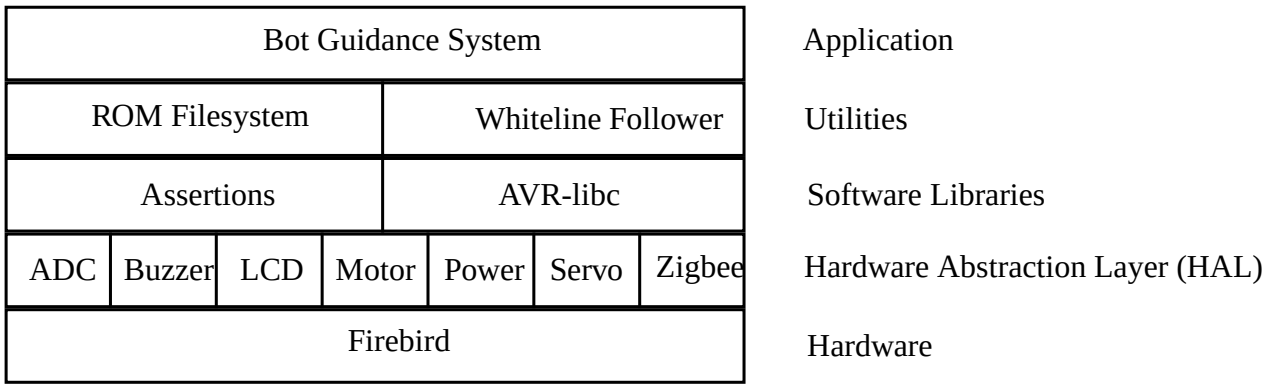


Figure 2: System Architecture of FireBird Bot

Task #1 : Move the Bot to desired trough with certain degree of accuracy using automatic bot guidance system

Task #2 : Fetch images from network camera and process it to detect fruits and cutter.

Task #3 : Control cutter and robotic arm to precisely cut fruits.

4.1 FireBird Bot System Architecture

Figure 2 shows architecture of system running on the Bot. We have followed modular design methodology.
There are multiple layers of modules, each supporting well defined primitives. Role of each module is summarized below:

- **Firebird:** This refers to hardware layer of FireBird Bot. Hardware is controlled by changing values in command and status registers. Refer to hardware [2] and software [3] manuals of FireBird V for further details.
- **HAL Layer:** Hardware abstraction layer (HAL) module hides hardware registers. It offers typical driver like primitives. `init<Device>()` API initializes the hardware of that device. Table 1 lists all HAL primitives and their use.
- **Assertions:** Assertion module mimics C assertions. It supports macro `ASSERT(condition)`. If assertion fails (*i.e.* condition is false), Bot halts, shows line number, source file name and failed statement on LCD screen and starts beeping until reset. Assertion

is quite useful for debugging. Define macro `NDEBUG` at the start of the file to turn assertions off. This replaces `ASSERT()` statements by empty statements.

- **ROM Filesystem:** This module mimics ROM based file system in limited manner. It redirects standard C file streams `stdin` and `stdout` to ZigBee. Standard I/O functions like `printf()`, `putchar()`, etc. send data over ZigBee to remote console. This simplifies data transmission over ZigBee as well as helps in debugging Bot code. `scanf()` and `getchar()` functions read from ZigBee. This allows to accept arbitrary data from remote machine. Bot can accept arena map files at runtime. AVR C library does not support files directly. We have provided support for compiled-in files using predefined file handles. File handle named `MAP_FILE` can be used to read from compiled-in read-only map file using standard buffered I/O functions like `fscanf(MAP_FILE, ...)`. More detailed discussion about file system and format of map file is included in file `readme.md` in the project source code.

- **White Line Follower:** This module supports white line related operations. It recognizes *checkpoints* (Explained in section 4.2). Table 2 explains supported operations. *Orientation* gives direction the Bot is facing towards. Bot supports only four possible orientations:

  - `EASTWARD`: Along positive X-axis
  - `NORTHWARD`: Along positive Y-axis
  - `WESTWARD`: Along negative X-axis
  - `SOUTHWARD`: Along negative Y-axis

4.2 Automatic Bot Guidance System

Problem of maneuvering the Bot to desired location in the Greenhouse is of fundamental in nature. Every greenhouse related project needs to solve it. We designed checkpoint based completely automatic bot guidance system (ABGS) which moves the Bot to desired location in the greenhouse accurately. Maximum error in moving the Bot is bounded by fixed constant and is independent of the time and distance covered.
ABGS is initialized with the map of the greenhouse. ABGS tracks the Bot using its Cartesian co-ordinates in millimeters. ABGS moves the Bot along white lines. It uses

Primitive	Use
`initAdc()`	Initialize ADC hardware
`getAdcValue(adc_channel)`	Read ADC value for specified channel
`initBuzzer()`	Initialize buzzer hardware
`buzzerOn()`	Turn on buzzer
`buzzerOff()`	Turn off buzzer
`initLcd()`	Initialize LCD hardware
`lcdHome()`	Place cursor at first column on LCD display
`lcdClear()`	Clear LCD screen
`lcdCursor(row, column)`	Place cursor at given row and column on LCD display
`lcdString(data)`	Write null terminated data on LCD display
`initMotor()`	Initialize DC motor hardware
`motorDirectionSet(direction)`	Controls direction of DC motors
`motorVelocitySet(lvel, rvel)`	Set motor velocity using pulse width modulation
`motorVelocityGet()`	Read motor velocity settings
`motorLeftPositionEncoder Init(lCallback)`	Register left positional encoder callback
`motorRightPositionEncoder Init(rCallback)`	Register right positional encoder callback
`motorLeftPositionEncoder InterruptConfig(state)`	Enable/disable left positional encoder interrupt
`motorRightPositionEncoder InterruptConfig(state)`	Enable/disable right positional encoder interrupt
`initPower()`	Initialize power management hardware
`powerOn(sensor_group)`	Turns power on for given group of sensors
`powerOff(sensor_group)`	Turns power off for given group of sensors
`initServo()`	Initialize servo motor hardware
`servoSet(motor, angle)`	Sets given angle for servo motor
`servoFree(motor)`	Unlocks servo motors
`initZigbee()`	Initialize ZigBee hardware
`zigbeeSendByte(u8Data, stream)`	Send one byte data over ZigBee
`zigbeeReceiveByte(stream)`	Receive one byte data over ZigBee

Table 1: FireBird HAL Primitives

Primitive	Use
`initWhiteLineFollower()`	Initialize white line follower module
`moveForwardFollwingLine ByDistance(distance)`	Moves along whiteline till specified distance (in millimeter) is covered
`moveForwardFollwingLine ByCheckpoint()`	Moves along whiteline until checkpoint is hit
`rotateBot(direction, angle)`	Rotates bot in specified direction by given degrees

Table 2: White Line Follower Primitives

position encoder to estimate current Bot location. Use of position encoders introduces error in tracking current bot location due to hardware imprecision. Such error in known as *location error*. Location error is bounded by using checkpoints. *Checkpoint* is a co-ordinate on the map whose location is *accurately* known. Whenever Bot moves over checkpoint it re-synchronizes its estimate of current location. Figure 3 shows automaton used for checkpoint synchronization.

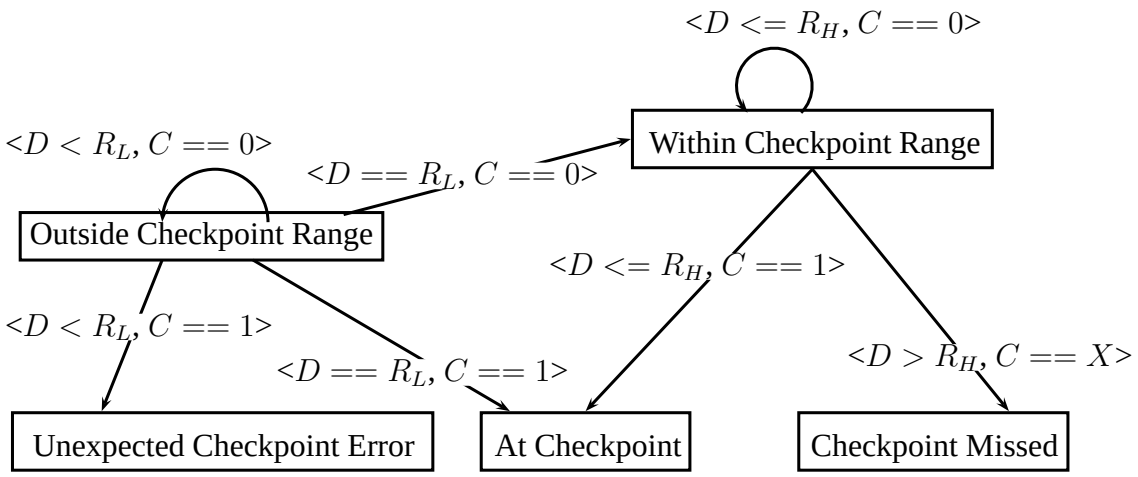
ABGS uses Floyd-Warshall all source shortest path algorithm over the greenhouse map. It supports `gotoPosition(x, y)` primitive which moves the Bot from current location to location with co-ordinates `(x, y)`. Table 3 lists primitives supported by ABGS.

Features of ABGS:

- Gives guarantee that *location error shall never exceed known constant bound*
- Precomputes all node shortest paths using Floyd-Warshall algorithm
- Uses integer only computations for speed
- Uses fast integer square roots
- Validated using assertions for manually specified loop invariants

4.3 Object Detection Using Computer Vision

We have used C-URL library to fetch images from wireless network camera on Linux machine. For image processing and object detection, we used OpenCV 2.4. Our primary



Legend:
D: Distance covered yet
R_L: Checkpoint range lower bound
R_H: Checkpoint range upper bound
C: Checkpoint sensing (1 means checkpoint hit, 0 otherwise)

Figure 3: Automaton for Checkpoint Synchronization

Primitive	**Use**
`initBotGuidanceSystem(fp, map)`	Initialize ABGS with from file handle `fp`
`gotoPosition(x, y)`	`x` and `y` are destination co-ordinates in millimeters
`setBotOrientation(orientation)`	Changes bot orientation
`gotoForward(distance)`	Moves the Bot forward by given distance in millimeters

Table 3: Automatic Bot Guidance System Primitives

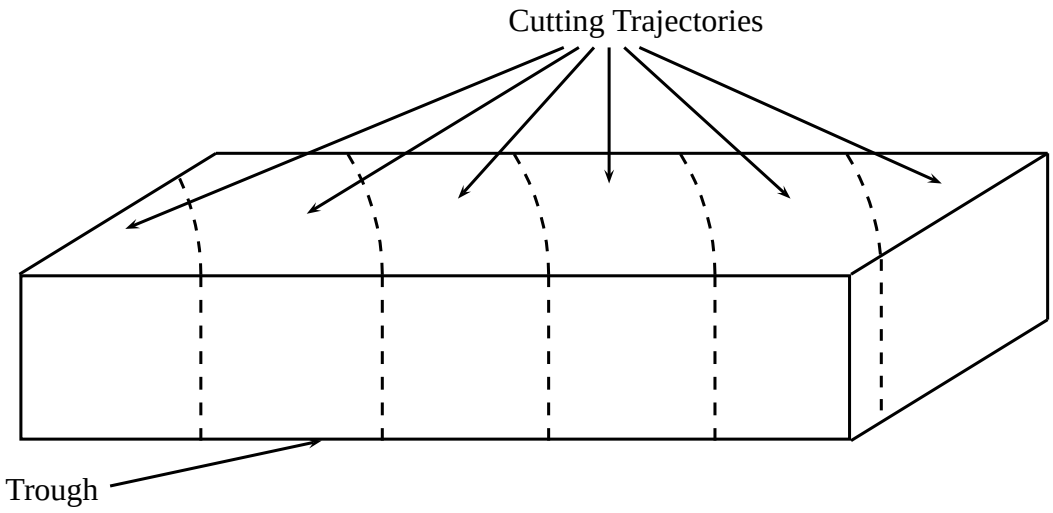


Figure 4: Cutting Trajectories for Single Trough

interest lies in detecting cutter and multiple ripened fruits. We pasted blue sticker on cutter and fruits are assumed to have reddish hue. Fruits should have certain minimum size. Such requirement serves two purposes. Firstly fruits that are ready for harvesting will be large enough. Secondly, it filters out spuriously detected objects making detection robust.

4.4 Fruit Cutting

Cutter fitted on robotic arm can move along both vertical and horizontal planes. Figure 4 shows cutting trajectories followed by robotic arm. Bot starts cutting from one edge and moves forward sweeping each trajectory. Center of fruit and cutter are aligned by moving robotic arm. When alignment matches, cutter jaws are closed to cut fruit. Fruit drops in collector bin attached to the Bot.

5 Testing Strategy and Data

We have developed testsuits for each of the modules Directory `FireBird/testsuit` contains all test code.

Battery Voltage	Battery API value	Meaning
9.0 V	> 900	Battery is sufficiently charged.
8.0 V	< 800	Battery is low Do not start new task. Finish current task.
7.5 V	< 750	Battery is critically low. Abandon current task. Turn off all servo motors Run towards recharge station.

Table 4: Battery Voltage Levels

5.1 Testing Automatic Bot Guidance System

We have followed method of *assertion based validation*. We have used assertions for sanity checks on all parameters and for enforcing loop invariant checks. Any logical error in the module would result in failed assertion. Then, we created test map of size 3000 mm X 3000 mm. We simulated `gotoPosition()` for all possible pairs of co-ordinates. Test procedures `test_gotoPosition()` and `test_gotoPosition2()` exhaustively test all possible combinations.

5.2 Testing Fruit Cutting

Size of fruit cannot exceed size of cutter jaw. We tested fruit sizes ranging from small beads to table tennis ball. We fined tuned colour parameters for small sized objects as small objects are difficult to detect.

5.3 Energy Consumption Statistics

We divided operational battery voltage range into three regions. Table 4 shows all significant levels and their associated semantics. Voltage values have been determined by repeated experimentation.
Energy consumption by harvesting task is shown in Table 5.

Action	Energy (Watt-Sec)	Energy (Watt-Hr)
Move Bot along whiteline	358 per meter	0.099 per meter
Cutter and arm movement + move forward by 10 cm	340	0.094
Scan sideways for fruits	1326	0.368
Cutting fruit	1029 per fruit	0.286 per fruit
One trajectory	2875	0.799
One trough (= 5 trajectories)	14375	3.993

Table 5: Energy Consumption for Harvesting Task

6 Design Challenges and Open Issues

Ensuring reliability and correctness of the project was the major design concern. We aimed for simple but effective Bot design. We chose modular design approach with each module providing well defined services to other modules. To ascertain about correctness, we used assertions which enforce invariants at critical points in the code. Any logical error would hopefully violate at least one of these assertions. We used white box testing for each module. Testing code is provided in `FireBird/testsuit` directory. Issues and challenges we faced are listed in following section.

Achieving Location Accuracy ABGS tracks current location of the Bot using wheel encoders. Hardware inaccuracies introduce small error with every movement. Errors accumulate over the time to significant proportion. To highlight specific problems, positional encoders do not reliably measure movement less than 4 mm. Both DC motors may not run with same speed for same value of PWM. There might be small difference in wheel circumference of each wheel. Response delay of each motor may be different. As result of these hardware inaccuracies, achieving same results with repeated trials is itself a challenge. We used multiple measures to overcome these issues. First we calibrated hardware and incorporated that data into distance calculations. We introduced checkpoints to resynchronise current location. Map of the arena is used to track current location and follow path to destination. This way ABGS is always aware of next checkpoint it should expect to hit in its path and how long is it from Bot's current location. Location error did not

exceed 7% (*i.e.* error up to 7 cm in while traveling one meter) in our experiments.

Mitigating Wheel Slipping We observed that robotic arm with the cutter at its front end imparts forward torque on the Bot. This causes center of gravity of the Bot to move towards front wheel. As a result, rolling friction between rear wheels and the ground reduces and rear wheels slip. Such slipping is undesirable as distance covered by the Bot no longer depends on the wheel rotations. Friction between tires and the ground can be increased by either by adding counter weights on rear side or by applying better tyre grip. Either approaches increase power consumption of the Bot. We mitigated wheel slipping by raising robotic arm to adjust center of gravity before moving forward.

Flow and Error Control for ZigBee ZigBee, being wireless communication medium, is lossy. We designed mechanism to detect errors and recover from them. We need flow control mechanism so that Bot receives commands at the rate which it can process. We designed our own protocol to meet these requirements. Data is sent over ZigBee using '!' terminated fixed sized data packets. Receiver replies with either positive or negative acknowledgement. Remember you cannot send raw binary data over ZigBee. You must first encode it into printable ASCII characters. Non-printable ASCII characters are reserved for serial link control commands. Although this approach serves our purpose, it has lower efficiency. Future projects should consider porting TCP/IP protocol stack on FireBird platform.

Map Specification ABGS needs description of the map. Map is supplied to ABGS in predefined format (explained in the code documentation). Map file is description of the graph annotated with geographical information for each node. Challenge was how to supply map file to ABGS as AVR C library does not suppot files. Thus we developed limited read-only file system to meet our purpose. There are two ways in which map file can be read. If map is known beforehand, map file can be compiled into hex code during development and accessed using pre-defined file handle `MAP_FILE`. Other way to access map file supports runtime loading of the map. We have redirected standard file streams to ZigBee. You can send map file from remote machine over ZigBee. Call map loading API `loadMap()` with first argument as `stdin`.

Power Management To minimize power requirement and extend battery time, we added support to turn off unused sensors. **Instructions from Firebird V software manual [3] do not work as intended for sensor power management.** We successfully turned off sharp IR range sensors 1 and 5.

Fruit Detection and Cutting Reliable fruit and cutter detection using OpenCV 2.4 was challenging. We had to pre-process image and fine tune colour parameters. Cutting needs correct positioning of the cutter and fruit. Cutting proceeds in two phases. In first phase, cutter is positioned approximately near the fruit. In second phase, cutter's position is fine tuned further along X and Y axis before cutting. Harvesting task runs as Linux application and sends commands to the Bot using ZigBee.

7 Future Work

We have identified following possible improvements for future projects.

Automatic Recovery of ABGS Current ABGS can only move Bot along white lines. This is primarily because ABGS loses Bot's position information if Bot deviates away from white line. Next challenge is drop white line requirement. Design method which would automatically recover Bot's current location when Bot is not on the white line.

Fruit Depth Information Single camera cannot give depth information of the fruit to be cut. Thus, we had to make conservative assumptions about fruit depth. Neither proximity nor range sensors are useful here as they fail to distinguish between fruit and leaves while gathering depth information.

Improve Arm Movements Current cutter design uses single arm. Fruit cutting throughput can be improved with complex robotic arm.

Improve Harvesting Our project is useful for harvesting small fruits like tomatoes, cherries or even flowers like roses. Future projects should considering harvesting for fruits like grapes, bananas and crops like corn, sugarcane, etc. Green colour of grapes make their harvesting challenging as detection is difficult.

8 Conclusion

This project demonstrates that it is possible to automate harvesting and monitoring in greenhouse. Although real environment of the greenhouse is rather different from the demonstration platform, this project should still function correctly. Choice of right abstractions make this possible. We substantiate this claim for white line. In real greenhouse, we cannot use white lines because of the soil and the dirt on the floor. However, white line for this project is mere abstraction of any mechanism that allows the Bot to move in straight line. We can use gyroscope or even laser guidance system to move along straight line in real greenhouse. This change merely needs modification at HAL layer code. Rest of the system remains the same.

Embedded systems suffer curse of hardware inaccuracies and physical faults. But, good design can overcome them. Predictability and accuracy can still be guaranteed within certain known bounds. Layered, module based design and right abstractions simplify development of complex systems.

References

[1] E-yantra website. `http://www.e-yantra.org`.

[2] Firebird v atmega2560 robotic research platform hardware manual. IIT Bombay & NEX Robotics Pvt. Ltd.

[3] Firebird v atmega2560 robotic research platform software manual. IIT Bombay & NEX Robotics Pvt. Ltd.