IIT BOMBAY

Software Requirements Specification

Gesture Controlled Robot

Using RTOS

D V Deepankar Reddy	09005060
M A Haseeb Ajmal	09005070
Gaurav S Chauhan	09005073
Ravi Vishwakarma	09005075

Index

1. Introduction

In this document we give a detailed description of the gesture controlled robot which we propose to design which includes purpose, features and also the constraints under which it must operate.

1.1 Definitions, Acronyms and Abbreviations

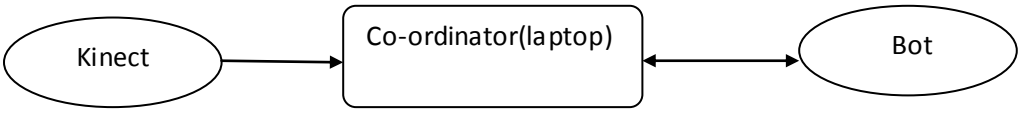
Term	Definition
Bot/Robot	The firebird V with ATMEGA 2560 which we are currently using in the laboratory
Kinect	The motion sensing input device developed by Microsoft
User	The person currently controlling the bot
Depth Data	The data that is produced by the depth image camera in the Kinect sensor. Each frame in the stream contains the distance, in millimeters, to the nearest object at a particular x and y coordinate in the depth sensor's field of view.
Skeletal Data	The data is provided to application code as a set of points, called skeleton positions, those compose a skeleton. This skeleton represents a user's current position and pose.
VGA	Video Graphic Accelerator
IR	Infra Red
API	Application Programming Interface
DirectX	It is a collection of API for handling multimedia related tasks, especially game programming, on Microsoft platforms.
Visual Studios	IDE for using the Software Development Kit
AVR Studios	IDE for c programming for the robot
µ-cos	The real time OS (RTOS) in which all the tasks are programmed
X-CTU	Software used to configure ZigBee wireless module.
IAR IDE	The IDE used for programming in µ-cos

1.2 References

- Getting Started with Kinect
- Sample Codes
- Getting Started with Visual Studios
- Another Guide for Kinect
- sample hand tracker
- Moodle resources for RTOS

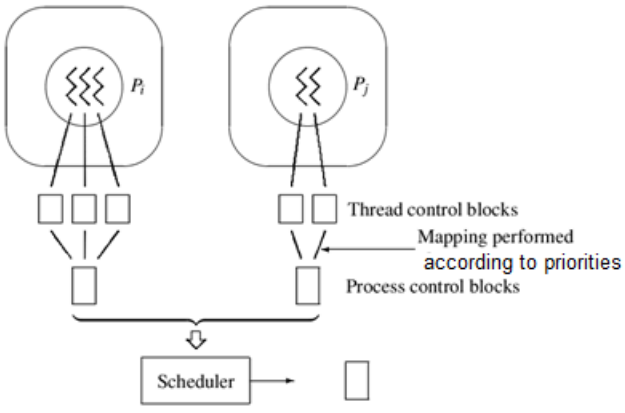
2. Overall Description

2.1 System Environment



The Robot has two active systems and one coordinating system. The User views the bot's environment from the coordinator and issues commands to it via the same. The communication between the coordinator and the robot is on a wireless frequency channel using zigbee communication. The User views the problems reported by the bot on the coordinator.

2.2 Usage of RTOS



firebird bot

Motivation: If raw code is written in c then adding new functionality or even a small extension to the project becomes extremely difficult since we need to figure out where to add/modify the code (i.e the exact location in the header or .c files is difficult to find). So for the sake of better re-usability of code we are planning to do the entire project in µ-cos.

Advantages: Doing this in RTOS gives many advantages including better readability and reusability of the code as in RTOS we make each functionality into a thread with some priority. Adding a new functionality just means creating a new thread and adding it to the stack of processes after allotting some priority to it.

So our main aim is to identify the key modules of the project which can be directly coded into processes and can be loaded onto the stack.

2.3 Product Perspective

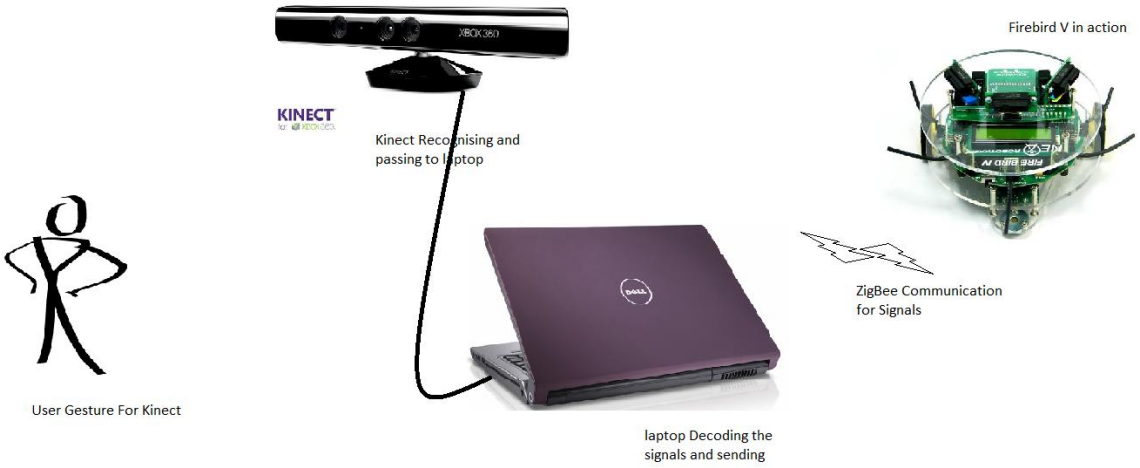
The user provides the input by movements representing gestures. These are captured by the kinect and translated to designer defined commands using Kinect API. These commands are now relayed to the bot using ZigBee communication. These are received by a particular task with high priority in the µ-cos operating system.

A camera could also be mounted on the bot if the bot and user aren't in the vicinity of each other and hence providing convenience to the user as to the location of the bot in it's environment.

2.4 Product Functions

Robot Movement:

Diagram:



Description:

a) A pre-recorded gesture is made by the user.
b) This gesture is captured the Kinect sensor.
c) If it matches any of the pre-existing gestures, it converts it into the corresponding command using Kinect API.
d) The bot receives the command sent by the co-ordinator over a wireless interfaces and the task in the µ-cos(which had been waiting for the command) is executed.
e) The robot now moves as required.
f) The environment is captured by the camera attached to the robot and sent to the coordinator for validation by the user.

2.5 User Characteristics

A gesture is any physical orientation of the user which can be detected by the kinect and used (indirectly) as a command for the firebird.

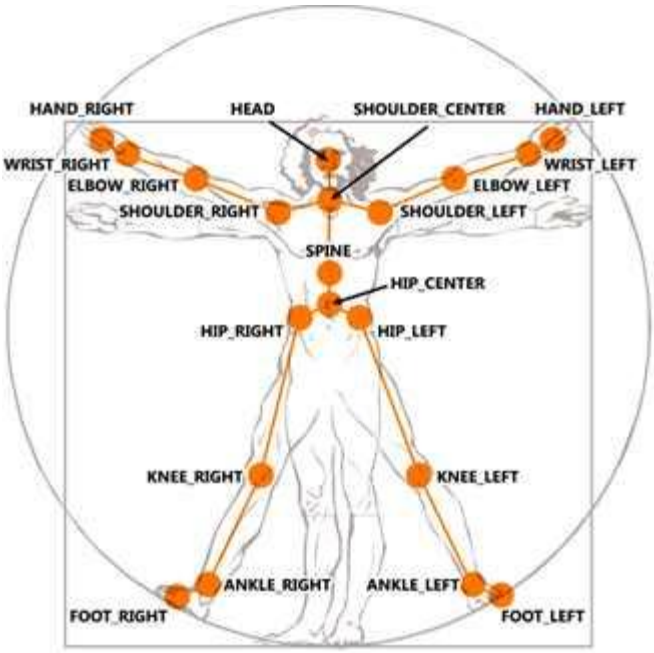
The gesture could be a "full-body" gesture or could consider only part of the body. For example, we could either consider the orientation (relative positions of various parts) of the entire body or take into consideration the orientation of only a particular part say a hand. In case we use full body gesture, we would have to do a lot more processing when compared to a single part. However we get a greater no. of gestures and hence greater functionality in this case when compared to the other.

The gestures could also be in 2D or 3D (in which case we would have to use depth mapping). The trade-off between these two would be the same as in the above case.

Kinect's View of a Gesture:

The kinect views a gesture as a set of joints and their relative positions. The joints which the kinect recognizes are shown in the figure below.

Hence we can define a wide range of gestures by simply defining the position of a subset/all of the skeletal joints.



We, in our project have decided to use arms and head which would be both convenient for the user as well as easy to code. The sample gestures shown here are only to give the reader an idea of what we wish to do:

As said above we plan on using both left and right arms and also the head. Thus we could control the motion of both the left and right motors independently.

The gestures made by the user should be vivid, distinct and as close to the list of gestures provided by the designer as possible. The intermediate gestures which occur as a part of transition from one to another shouldn't cause any problem.

The success/failure of robot motion as a result of the gesture can be confirmed by the scan of the environment obtained by the coordinator from the camera attached to the bot.

2.6 Constraints

1. The Kinect SDK is supposed to be used on MS Windows 7 or higher versions with DirectX SDK. Hence this restriction applies for the software being developed.

2. The coordinator should have a good processor and VGA card for proper functioning of the Kinect Sensor Device.

3. The distance between the user and Kinect Interface must be 4 to 5 meters. This is done to prevent any interference from influencing the function of the Sensor.

4. The distance over which Zigbee communication is possible is also restricted.

5. The bandwidth of the Zigbee communication should be sufficient to manage the payload i.e. the commands sent over the interface to the bot and video transmitted to the coordinator by the bot.

6. The rate at which the gestures can be accepted is restricted by the processing speed and propagation delay.

2.7 Assumptions and Dependencies

- There should be no disturbance between the sensor and user. Any form of disturbance might result in sensor resetting itself.

2.8 Requirement Subsets

1. Gesture Recognition
2. Transmission of Commands
3. Interpretation of Commands
4. Performing relevant task in u-cos
5. Environment scan

3. Details

3.1 Functionality

The main motivation behind this project was to make machines do the work which humans do with very little energy input from the human but high productivity. Our project is only a fundamental step towards achieving this aim. So we wish to provide as many basic functionalities as possible.

More specifically, the need for a remote to control a robot is annulled and replaced by gestures which could be grasped by any layman given a manual with figures to explain the gesture-motion mapping. The usability of the system is improved (when compared to a hand

holding device) . This could also be a step towards modern video games which simulate the actions of the hero in the game as per the gestures of the user.

The wireless network provides point to point connectivity reducing the possibility of interference on the communication channel. The gestures are converted to commands and fed to the bot hence receiving required responses from the bot. The problems faced by the bot can be reported as a set of preformatted responses hence getting an idea of the environment in which the bot is present. We could alternatively use the live feed from the camera mounted on the bot.

Usage of µ-cos is mainly divided into identifying the key modules of the project which can be directly coded into processes and can be loaded onto the stack and assigning proper priorities. The modules should be divided in such a way that there is no overlap between any two modules (i.e. they should be completely independent).

The assignment of priorities to tasks should be done with utmost care. For example adaptive cruise control should be given higher priority when compared to the gesture simulated movement. One reasoning behind this could be that, if say we encounter an obstacle and simultaneously receive inputs from the user it would be useless to perform the actions of the user since it would be effectively nullified by the presence of the obstacle. Hence if the obstacle situation is handled first, then the user actions make some sense.

3.2 Supportability

3.2.1 Basis for future work

As can be seen from the motivation provided in the above section, there is a lot of scope for building upon this project. In fact this project is itself an improvement to the pre-existing project (converting to u-cos and providing additional features). Future groups could add more and more functionality to this project to approximate it to function as a human servant. The gestures could also provide more complex actions (like performing a series of actions to get some work done)

Speech recognition can also be done to control the bot as kinect as four microphones .The language libraries can be used to match the spoken words to interpreted words.

The gesture library we are currently intending to develop can be improved a lot.

3.2.2 Modules

1. Kinect programming using Microsoft APIs for Natural User Interface for gesture recognition

2. ZigBee communication module for wireless communication

3. µ-cos interface for signal generation we may also use esterel if there are some problems encountered with µ cos.

4. A module for gestures which can potentially be used in any other project independently. This module can be extended to form a gesture library for kinect.

Any of these modules can potentially be used in other projects requiring them. The entire RTOS can be used for further development and addition of features.

3.2.3 Documentation

A full-fledged documentation would be provided in the form of comments in the code for ease of understanding for future groups.

3.3 Design Constraints

The hardware restrictions which limit the functionality of the bot are:

1. Due to absence of motor mount for the camera the bot needs a full rotation to get complete view of the environment rather than the camera being rotated.
2. Use of Visual Studios is forced due to the requirements of Kinect SDK and DirectX SDK.

3.4 Interfaces

3.4.1 User Interfaces

The part of the system which provides input output interface for the user is the user interface. It consists of the following:

- Interface of the user with the kinect which includes the skeletal view, the camera view and IR grayscale distance measurement view of user.
- Display of the gesture as captured by the kinect.
- Display of the environment scanned by the camera on the robot.

3.4.2 Hardware Interfaces

These essentially are the hardware equipment required for the application to run on.

- The Microsoft Kinect Sensor to take the input (gestures) from the user.
- VGA Card with DirectX support for the laptop.
- Camera on the robot with data bus connectivity to ZigBee module.

3.4.3 Software Interfaces

These are the IDEs and SDKs required for software development.

- Microsoft Kinect SDK
- DirectX SDK
- Visual Studios
- Atmel AVR Studio

3.4.4 Communication Interfaces

These are the interfaces required to communicate between robot and coordinator.

- ZigBee Wireless Modules
- X-CTU ZigBee Configuration

4. Quality control

4.1 Recognizing Hand Gestures

All approximations to a gesture should result in the same command being issued to the bot. We'll be using very distinct gestures and having threshold values for the positions of the various joints which would be used for gesture recognition.

We, in our project have decided to use arms and head which would be both convenient for the user as well as easy to code. The basic idea of our gestures had been made clear in section 2.5.

4.2 Adaptive Cruise Control

We plan to do this in order to avoid the obstacles in the environment of the robot

4.3 Conflict Resolution

When commands are issued in a haphazard manner the robot is to be programmed to manage such situations

5 Risk Management

1. **Gesture Recognition** : The gesture to pre-recorded gesture matching should not be too strict. It should be lenient enough to match close enough gestures. However it should not misinterpret any casual movement to pre-recorded gestures. Thus trial

runs are required to ensure that the user gets accustomed to the pre-recorded gestures.

- ***Fall Back Plan :*** try to write code to identify general gestures and declare them void. We will be conducting some experiments which could lead to sampling of gestures and hence can decrease the scope of this problem.

2. **Gesture and Signal Mapping**: There can be delay in decoding the gesture signals and sending them back to the bot and this delay can cause discrete rather than continous movement and this can cause the micro-controller problems due to sudden rise and fall of voltage.
    - ***Fall Back Plan:*** A signal to perform a previous gesture is emmited till a new gesture is recognized
3. **Speech Recognition problems**: Speech patterns and phonetics and individuals differ from one another especially in a team of 4 if implemented.This is a problem only if the speech interface is implemented
    - ***Fall Back Plan:*** make only one person give the demo.try to make a interface for training of the signals
4. **Camera feed problems:** Transmission of camera feed from camera can cause many problems such as the link for transmission of the feed and the bandwidth insufficiency and the corruption of transmitted packets.
    - ***Fall Back Plan:*** Can send encrypted signals indicating the objects in front and the objects in the side using the various sensors and hence can provide a comprehensive view of the environment
5. **RTOS:** There can be serious synchronization problems while using RTOS. In such cases we would be combining lot of small tasks to form a bigger one which could lead to lesser problems.