INSTITUTE OF SCIENCES AND ENGINEERING

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

AN ENCODER FAULT TOLERANT FPGA BASED ROBOT CONTROL USING BLUETOOTH OF A SMART PHONE

A Thesis

submitted by

Muhammad Ameen Majeed

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

April 2017

Program: Power Electronics and Clean Energy Systems

AN ENCODER FAULT TOLERANT FPGA BASED ROBOT CONTROL USING BLUETOOTH OF A SMART PHONE

A Thesis

by

MUHAMMAD AMEEN MAJEED

submitted to the Institute of Sciences and Engineering of

OKAN UNIVERSITY

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair
Asst. Prof. Dr. S. Barış ÖZTÜRK
Supervisor

Prof. Dr. R. Nejat TUNCAY
Member

Assoc. Prof. Dr. Özgür ÜSTÜN
Member

April 2017

Program: Power Electronics and Clean Energy Systems

ABSTRACT

AN ENCODER FAULT TOLERANT FPGA BASED ROBOT CONTROL USING BLUETOOTH OF A SMART PHONE

An FPGA based Bluetooth controlled robot with encoder fault tolerant algorithm is presented. The smart phone is used to obtain user intentions, such as turning, speeding or braking and this data is sent to a robot using Bluetooth. The motor control algorithms and robot communication interfaces are implemented on FPGA for parallel processing. The robot is capable of changing its position with the help of two independent PI closed loop speed controlled DC motors. The motor speed is determined by smart phone accelerometer's sensor data. The FPGA receives speed information and generates PWM signals based on this speed data. Motor rpm is calculated via rotary encoder's feedback. In case of an encoder failure an onboard gyroscope helps maintaining normal operation. An android based smart phone application has been developed. MATLAB is used for simulating an encoder failure and observation of results. The motor control algorithm has been implemented using Verilog and tested on the field.

Keywords: PWM; Bluetooth, PC, DC Motor; PI controller; Rotary encoder; Moving average Filter, Android smartphone, PmodBT, PmodHB5, PmodUSBUART, accelerometer, MATLAB, Gyroscope, Sensor fusion, Mobile Robot.

KISA ÖZET

ENKODER HATASI TOLERANSLI FPGA BAZLI AKILLI TELEFON İLE BLUETOOTH KONTROLLÜ ROBOT

Enkoder hatasına toleranslı algoritmaya sahip olan, FPGA tabanlı Bluetooth kontrollü bir robot sunulmaktadır. Dönme, hızlanma veya frenleme gibi kullanıcı hatalarını ortaya koymak için akıllı telefon kullanıldı ve bu veriler Bluetooth kullanılarak robota gönderildi. Motor kontrol algoritmaları ve robot komünikasyon ara yüzü gibi paralel işlemler için FPGA'de uygulandı. Robot, iki bağımsız PI kapalı çevrim hız kontrollü DC motor yardımıyla pozisyonunu değiştirebiliyor. Motor hızı, akıllı telefondaki akselometre sensörü verileri tarafından belirlendi. FPGA, hız bilgisini alıp, bu hız bilgisi doğrultusunda PWM sinyalleri üretir. Motor devri, döner enkoder geri beslemesi ile hesaplanır. Enkoder arızası olduğu durumda, yerleşik bir jiroskop, robotun normal çalışmayı sürdürmesine yardımcı olur. Android tabanlı akıllı telefon uygulaması geliştirildi. MATLAB, bir enkoder hatasını simüle etmek ve sonuçları görmek için kullanıldı. Motor kontrol algoritması Verilog kullanılarak gerçekleştirildi ve alanda test edildi.

To my Parents

ACKNOWLEDGMENT

All praise belongs to Almighty Allah. I express my gratitude to my advisor Asst. Prof. Dr. Salih Baris Ozturk for the continuous support of my M.Sc. study and research, for his patience and motivation. His guidance helped me in all the time of research and writing of this thesis.

Besides my advisor, I would like to thank the rest of my thesis committee for their encouragement, comments and queries.

I would like to thank my friends for their support and encouragement. Last but not the least, I would like to thank my family, particularly my parents and to my grandma, sister and brothers for supporting me with their limitless love, care, attention and whatever they could have done throughout writing this thesis and my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

FPGA	Field Programmable Gate Array
PWM	Pulse Width Modulation
PID	Proportional Integral Derivative
PI	Proportional Integral
MEMS	Micro Electro Mechanical Systems
UART	Universal Asynchronous Receiver Transmitter
SPI	Serial Parallel Interface
USB	Universal Serial Bus
RPM	Rounds per Minute
CLK	Clock
SCLK	Synchronous Clock
Tx	Transmit
Rx	Receive
DPS	Degrees per Second
MOSI	Master Out Slave In
MISO	Master in Slave Out
DC	Direct Current
DSP	Digital Signal Processor
CV	Computer Vision

I. INTRODUCTION

1.1. Smart Phone Technology and Mobile Robots

Smart phone technology has been evolving and its adoption continues to grow [1], [2]. Latest smart phone processing capabilities are comparable to a low end personal computer [3]. Presence of wireless communication capabilities and on board sensors, has allowed smart phones to be used in a significant number of remote applications [4], [5], [6].

Smart phone interfaced robots have brought a new dimension to the use of smart phones as a control device for various applications such as drones, search and rescue, surveillance, educational and indoor robots [7], [8], [9]. Although smart phone processes sent and received data, the robot must perform various tasks, such as motor control, perform predetermined operation in case of communication break, without smart phone intervention. Since robot must complete these parallel tasks, FPGA is a good candidate for these applications.

1.2. Problems with Encoders as Rotor Speed Sensors

Encoders are widely used to obtain rotor speed and position information in closed loop motor control systems. A faulty or a failed encoder at the robot's motor may lead to incorrect speed information and result in collapse of the encoder feedback [10] based closed loop controller. Encoders are prone to failure for a number of reasons as discussed in [11].

An Encoder failure, Surface roughness, wheel slip, tread wear, an obstacle or an external push can infer a deviation in course of mobile robots with only encoder feedback controlled motors. A need for a method for calculating angle deviation

independent of encoders and correcting the direction of the robot is eminent. In this work, the use of computationally simple and fast acting algorithm in compensating encoder error is studied.

1.3. The FPGA Technology

FPGAs are Integrated Circuits that allow implementation of custom defined digital circuits after the device is manufactured. These designs can be updated or changed as needed in the field. FPGAs contain a matrix of programmable logic cells and configurable switches that allow the logic cells to be wired together in different desired configurations.

A custom design can be realized by configuring the logic cells and making them perform a specific function while and a set of configured logic cells are interconnected via the programmable switches to implement desired digital circuit. In most FPGAs logic blocks can also function as memory elements.

Other than this reconfigurable circuitry the manufacturer may put dedicated hardware circuits for performing specific tasks. These circuits are known as IP Cores and usually consist of hardware circuits that are commonly used, such as protocol controllers, memory blocks and memory controllers.

In contrast with ASICs, FPGAs allow you to realize exactly the hardware that is tailored to the need and that design (contrary to ASIC designs) is too changeable if required. FPGA based designs are often quicker to realize than ASIC designs. In some cases, their performance and efficiency is comparable to their ASIC counterparts. FPGAs are also used for prototyping ASIC based designs before their final version is manufactured. FPGAs are capable of parallel execution of processes which allows the

tasks that need to be done in repeated cycles much more efficiently and quickly than microcontrollers.

Applications include but are not limited to DSP, SDR, ASIC prototyping, Medical Imaging, CV, Speech recognition, Cryptography, Bioinformatics, Computer Hardware Emulation and Radio Astronomy.

1.4. Introduction to MEMS

MEMS (Micro Electro Mechanical System) is a miniature machine that has both mechanical and electronic components. The physical dimension of MEMS can range from several millimeters to less than one micrometer. MEMS devices vary from simple devices with no moving elements at all to complex electromechanical systems. Even if there are no moving mechanical elements but the device possesses mechanical characteristics, it fulfills the criteria to be defined as MEMS. Some MEMS devices convert a mechanical signal to an electrical signal and function as a transducer.

A wide array of MEMS based sensors (transducers) is developed. Well known ones are inertial sensors, temperature, pressure and magnetic field sensors to name a few. MEMS are manufactured using same batch fabrication techniques used to manufacture ICs. Many commercial MEMS products are integrated and packaged together with microelectronic ICs.

Nomenclature for MEMS differs from region to region for e.g. in Japan they are more commonly known as microsystems and in Europe they are commonly referred to as Microsystems Technology (MST).

1.5. Employability of Inertial Sensors for Angle Estimation

Inertial sensors are used in a no. of applications alongside conventional encoders to improve speed calculations or angle estimations. A Method for angle estimation in industrial manipulators using only inertial sensors is discussed in [6]. A sensor fusion approach (encoders and IMUs) for angle estimation used to improve precision and reduce cost is presented in [12] and [13].

Human body motion kinetic modeling [10], GAIT based human modeling, walk assist systems [14] and rehabilitation therapy systems [7] readily employ inertial sensors for angle estimations.

1.6. Thesis Outline

The contribution of this work is FPGA implementation and testing of a gyroscope feedback based control loop on a smart phone's Bluetooth controlled robot which resists set path deviation and retains normal operation in a scenario where either the primary encoder sensor fails or an external disturbance attempts to deviate the robot from its set path. In this research, a 3-axis MEMS gyroscope sensor is applied for angular movement estimation and MATLAB is used for providing test stimuli to the robot and capturing results.

The rest of the document is organized as follows. In Section II "Methodology", brief information about the system's hardware, used software tools, realized FPGA circuit, android application and MATLAB code is presented. The FPGA circuit along with its constituent modules is discussed in Section III "Realized FPGA Circuit". Implementation and measurement results are given in Section IV "Experimental Results". Concluding remarks are given in Section V "Conclusion". This is followed

by two Appendixes which present MATLAB and Verilog code snippets used in this project.

II. METHODOLOGY

2.1. System Introduction

This project consists of three main elements,

1- The development of a Bluetooth based smart phone interface to allow control of the vehicle using an android application.

2- An encoder fault tolerant motor control algorithm employing a Gyroscope as a secondary sensor.

3- Development of a MATLAB based system for providing test stimuli and observation of results.

The system is based on an FPGA based digital circuit which is capable of establishing a Bluetooth connection between an Android based smart phone or a PC. Encoder feedback is used as primary feedback source for motor speed control loop. An encoder fault tolerant motor control algorithm based on a secondary gyroscope sensor feedback based control loop is realized and tested.

The rpm speed is derived from smart phone's tilt angle which is calculated using the smart phone's accelerometer sensor data. A colored circle to indicate the tilt of the phone is displayed on the screen as a visual aid. The smart phone transmits the desired speed in rpm over the Bluetooth link to the FPGA. The FPGA maintains two DC motors at the desired speed with the help of PI closed loop controllers using Hall Effect rotary encoders and a gyroscope.

In case of an encoder failure the rpm values are not calculated correctly and the encoder feedback based control loop fails. As a result, the robot deviates from its set

path. This deviation in path is detected by the on-board gyroscope. The gyroscope feedback based controller kicks in and helps maintaining system operation.

For simulating an encoder failure and observing results a MATLAB code has been developed. With the help of this code various FPGA circuit parameters can be set and variables can be observed on a PC in real time via Bluetooth connection.

There are two modes in which the robot can be operated. The robot can be operated in only one mode at a time. If the robot is connected to a smart phone, no data is sent from the FPGA to the smartphone and the circuit only receives the rpm speed information for the two motors. However, if the robot is connected to MATLAB via a PC, various parameters for the FPGA circuit can be set along with sending rpm speed and motion angle information. The FPGA circuit on a request from the MATLAB can send data to the robot as well. The data usually consists of FPGA circuit's variables that are desired to be monitored.

2.2. Employed Hardware Equipment and Devices

All of the Hardware equipment and FPGA kit, sensor modules, Motor driver and direction control circuits used in the system are from Diligent. All the Electronic Hardware Modules and the FPGA kit are compatible to one another.

2.2.1. Basys2 FPGA Kit

Basys2 FPGA kit (250K gates model) from Diligent is used as the FPGA hardware platform. The kit has sufficient Hardware resources to support our system. The kit employs a Spartan 3E-250 CP132 FPGA from Xilinx. The FPGA has 250K gates in total (out of which almost 38% have been employed). The Basys2 kit comes as two models, 100K gates and 250K gates models. Previously a 100K gate model kit was used

to implement the system but its resources were not enough to support the system. Hence the 250K gates model Basys2 FPGA kit is used. The kit uses a USB port which allows powering the device and easy programming and data transfer between the PC and the kit. A battery connector allows external power source to be connected to the kit if powering the device via the USB port is not desired. Each time the FPGA is reset or a power disruption occurs, the contents of the loaded program file that define the FPGA circuit (bit file) are lost and the FPGA is needed to be programed again. A non-volatile Flash ROM can be used to store and load a project bit file on every system startup or reset. Important hardware components on the kit include 8 LEDs, 4 digit seven segment displays, 4 buttons and 8 slide switches. A 50 MHz on board clock is used as the primary clock source (selectable among 25, 50 and 100 MHz). Four I/0 ports allow the kit to be connected with the external devices.

2.2.2. Diligent PmodBT

PmodBt from Diligent is used as the Bluetooth device for our system. This device houses a LMX9838 Bluetooth integrated system IC from Texas Instruments. This Bluetooth device uses UART interface for host communication.

The PmodBT's UART baud rate is selectable via jumpers present on it. The baud rate is set at 115.2 kbps. Some basic parameters at PmodBT are needed to be known or are required to be reset. These include device name and device PIN code. These parameters are reset by connecting the PmodBT to a PC with the help of a PmodUSBUART device. A computer program from TI called "Simply Blue Commander" is used to reset these parameters. For connecting the PmodBT to the PmodUSBUART a signal rerouting circuit is synthesized on the FPGA. The circuit reroutes UART data to a PmodUSBUART and vice versa.

2.2.3. Diligent Pmod Gyro

The PmodGyro houses a L3G4200D STMicroelectronics MEMS gyroscope. This device can utilize two communication interfaces I2C and SPI. In this system only SPI is used. The L3G4200D can measure angular rate of change in three axes. Each axis value is 16 bits wide. Angular rate of change value is selectable from 250, 500 and 2000 dps. Our system uses 250 dps angular rate of change which gives the highest precision that is 8.75 mdps per digit.

2.2.4. Diligent Motor Robot Kit Basic

The Diligent Motor Robot kit consists of parts belonging to a robotic development kit. These parts include a steel platform, a pair of wheels, a pair of DC motors, necessary supports, connecting wires, nuts and bolts. This kit is the platform for all the hardware mounted on the remote vehicle. The Basys2 FPGA kit and the other hardware used are mounted on it.

2.2.5. Diligent PmodHb5

PmodHB5 modules are used to drive and direction control the DC motors. The module employs a full H bridge circuit (2A-12V) for direction control. The 6 pin port (J1) has two encoder sensor feedback pins, a pair of supply, enable pin and direction control pin. The PWM duty cycle variation is provided at enable pin. Connector “J3” and “J2” are used to provide power and connect to the motor respectively.

2.3. Software Tools

For Verilog code development Xilinx ISE V 14.5 is used. All the FPGA circuit modules have been coded using Verilog HDL. Adept from Diligent is used for programming the binary file to the FPGA.

The Android application has been coded using java. However, some of the user interface components are coded using XML. Android SDK tools are used for Android application development. The application is tested on a device running Android OS version 4.0, however it is compatible with devices running Android OS version 2.2 and above.

The MATLAB code is developed using MATLAB 2015a.

2.4. System Design

The android application connects to the robot via Bluetooth and transmits motor speed in rpm and direction information to the FPGA. The raw accelerometer values are adjusted by the application, so that they represent motor speeds in rpm. An on screen moving circle is displayed which gives visual aid to phone tilt.

The FPGA circuit receives and processes Bluetooth data. Simultaneously the circuit processing data from the encoders calculates current DC motor speed in rpm. A PI controller adjusts a 16 bit PWM signal according to the calculated rpm. Direction is set according to the received direction information. The motor direction switching has enough delay to prevent any damage to the motor driving circuits.

2.5. Robot's On-Board Hardware Arrangement

The Basys2 kit is mounted on a robotic platform called Diligent motor robotic kit. Four 6 pin I/O ports are available on the Basys2 Kit. The PmodBT (Bluetooth module) is connected at port A, port B and D are connected to the two PmodHB5s (motor driver circuits) via a connector cable to the left and right hand side, the PmodGyro (Gyroscope sensor) is mounted on port C.

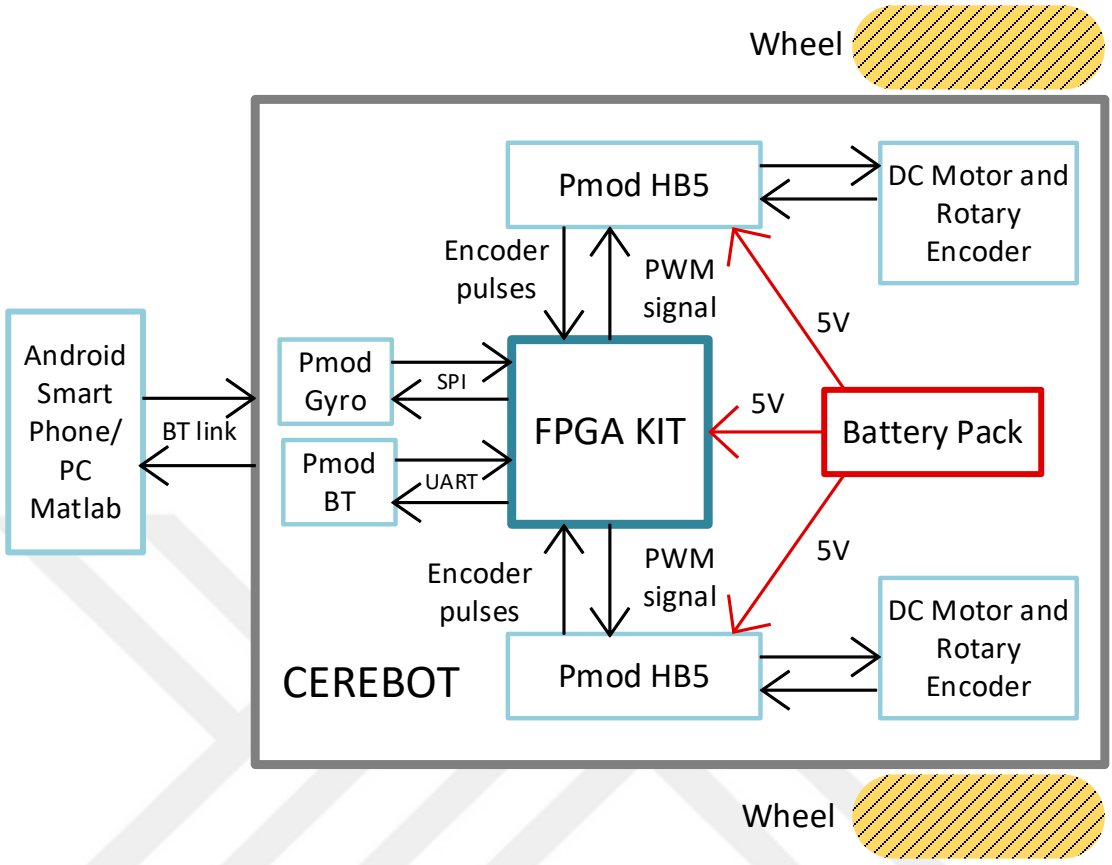

Figure II.I. Robot's Hardware Arrangement Block Diagram.

Two Battery packs are secured at the back of the robot. One battery pack is used to power the FPGA and the other to power the motors. A small connector at the front left side of the robot connects the battery pack to the two PModHb5s which deliver the power to the two motors. The PmodHb5s and the motors are mounted under the robot. Please refer to Figure II.II for more details.

The PmodBT has a 12 Pin connector. These pins are divided into two 6 pin sets. The set from pin 1 to 6 contains all the required pins for necessary device operation. The other set from 7 to 12 has three NC (No Connection), a duplicate set of power supply and a reset pin. This set is left unconnected to the FPGA kit at a tradeoff for reset functionality. The PmodHB5 has three ports. Two ports have six pins and one has two pins. One of the six pin ports are connected to the FPGA.

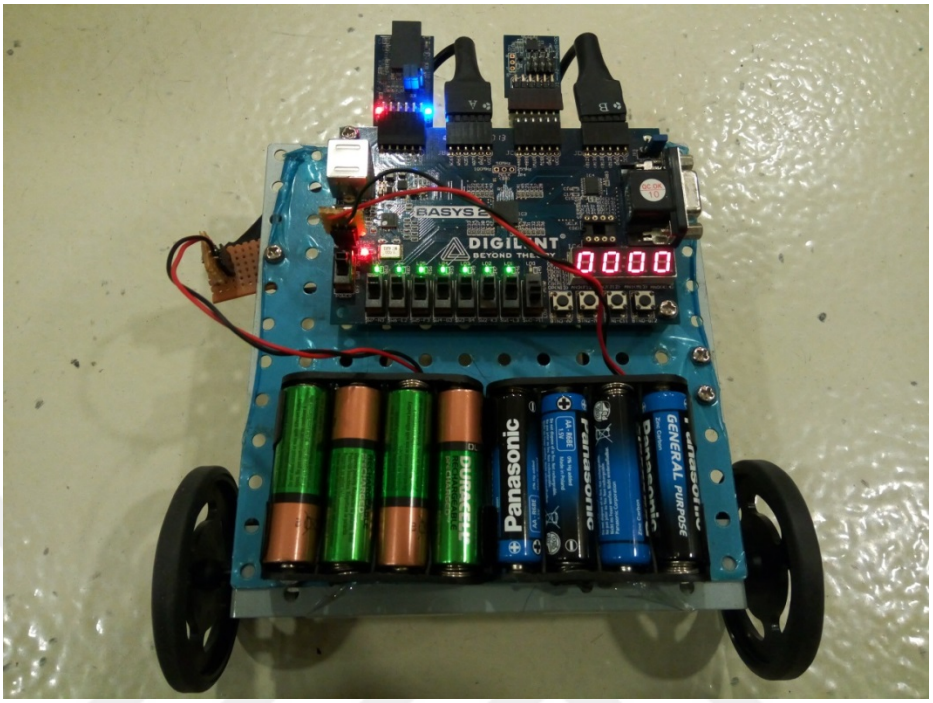

Figure II.II. Robot's Hardware Setup.

The FPGA connected port has a set of power supply pins, a pin to receive motor direction a pin to receive enable signal and two pins to feed encoder data to the FPGA. The other port is connected to the motor, a set of power supply pins, a set of two pins which receive encoder feedback and a set of power rerouting pins for motor direction control. The two-pin port is connected to the battery. The encoders are attached with the motors.

The PmodGyro has two headers one 12 pin and the other 8 pin. The device can communicate via SPI or I2C interfaces. Header J1 and J2 are used for SPI and I2C communication respectively. Only the SPI interface circuit has been realized on the FPGA the latter is not used. Hence the 8 pin header is not used and remains unconnected. The 12 pin SPI header is divided into two 6 pin sets.

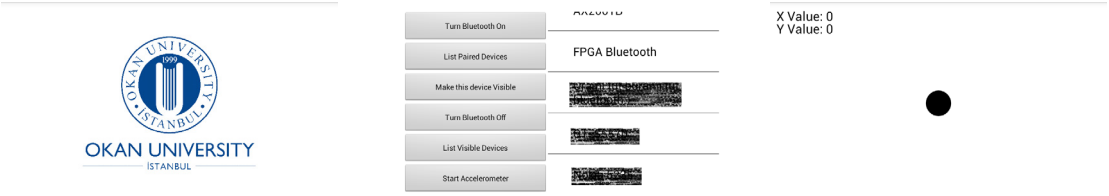

Figure II.III. Android Application Screen Shots.

The set having pins from 1 to 6 provides all the necessary functionality for SPI communication. The other set has power supply pair pins, two unconnected pins and two interrupt pins. The interrupt functionality is not used and is left unconnected. Please refer to Figure II.II for details.

2.6. Android Application

An android application is developed solely for this system. This application connects to the remote PmodBT Bluetooth device and sends motor speed and direction information to the FPGA circuit. The communication between the smartphone and the remote Bluetooth device is unidirectional. The smart phone only sends data to and retrieves no information from the remote device.

The application consists of two activities “MainActivity” and “BluetoothActivity”. The main activity is the default activity and is the first one to run at application startup. This activity displays Okan University’s emblem and brief information about the application. This activity’s Screen View is handled by “Title View” class.

A tap anywhere on the screen results in start of the BluetoothActivity. The java class that handles activity’s screen view is named “BluetoothAcceleromoterView”. This activity fetches the accelerometer data and enables Bluetooth communication. All Bluetooth operations are handled in a separate thread named “ConnectThread”. The

Activity can turn the Bluetooth Device on or off, fetch and list existing paired devices and connect to a Bluetooth device. The raw accelerometer sensor data is tailored to be sent to and used by the FPGA circuit by this activity.

The view affiliated with this activity displays buttons, to turn Bluetooth on or off, list already paired devices, make the smart phone's Bluetooth visible to other devices, list nearby visible devices and start the accelerometer. If the "List Paired Devices" or "List Visible Devices" Button is pressed a list of paired or visible devices appears on the screen. The smartphone's Bluetooth is connected to the remote device by taping its name in the list. A short toast informs the user if the Bluetooth link is successfully established or the attempt is failed.

Once a Bluetooth link is established, accelerometer based motor speed and direction information is sent by pressing the "Start Accelerometer" Button. A new set of information is sent every 100 ms. The screen View changes and displays current speed in rpm values that are being sent. A circle moving with respect to orientation provides a visual aid in controlling the remote device.

2.7. MATLAB Code

MATLAB Instrument control toolbox provides support for Bluetooth communication and is used to connect, send and receive data to and from the robot. The data sent to the FPGA includes motor speed rpm set points, controller gains, angular displacement set points and control commands to request data from the FPGA circuit, enable/disable encoder's signals and controller circuits.

Upon data send request the FPGA circuit sends a data set of 16 bytes. Variables from any module can be accommodated in this data set in the FPGA circuit and sent to MATLAB for observation. Variables that change at a frequency less than 800 Hz are

observed without any information loss. Variables with higher frequency of change are also observable but an information loss is eminent.

A set of codes is defined to communicate with the remote Bluetooth device. Functions and usage of this code set is defined in Table III.1, Table III.2 and briefly discussed in Section 3.4.

A Bluetooth object is created with remote Bluetooth device's name and channel information. Once a Bluetooth object is constructed the execution of MATLAB "fopen" function with the Bluetooth device as its argument attempts to connect the MATLAB Instrument Control Toolbox to the remote Bluetooth device.

As an example, the code snippet for an experiment discussed in Section 4.2.4 is provided in Appendix B. A set of variables is initialized to collect data from the remote device. Please refer to the MATLAB code in Appendix B, at line 13 and 17 motor A and motor B rpm speeds are set by writing their respective codes to the remote device. This is followed by setting the proportional and integral gains for the closed loop controllers (lines 21 to 47). A for loop then loops for 4 seconds. on each iteration, which lasts 100 ms the code inside the loop requests, collects and displays information from the remote device in real time. The MATLAB functions "tic" and "toc" at lines 54 and 58 help maintain precision in data collection interval delay. The time MATLAB takes to process the incoming data and display it on the screen varies with almost each loop iteration. Hence the time MATLAB takes to complete the process is deducted from the time it has to halt for 100ms. Data is read via data read function at line 64. The function for displaying the data is executed at 67.

Gyroscope angular set point is set at lines 79 to 85. The results are retrieved for the next seven seconds. The code is the same as lines 53 to 68 and is omitted from the

appendix. The Gyroscope angular set point is once again set at 91 to 97 and results are gathered. Before releasing the Bluetooth link some parameters are set to 0 to stop the robot. The Bluetooth link is then released and a relevant message is displayed on the MATLAB command window.

2.8. PmodBT Communication Modes

PmodBT Houses an LMX9838 module. UART communication interface is used between the host and the device. The device uses two modes in UART communication namely Command Mode and Transparent Mode.

2.9. TI LMX98938 Command Mode

In command mode, all data sent to the device at the UART is tried to be interpreted to known commands. These commands are used for setting various device parameters or for commanding the device to do a certain task, for example setting Device name and pin code, commanding the device to connect to a remote device,
UART baud rate settings etc. Under default device startup settings, the device is in command mode on startup, provided that it is not connected to any remote device.

The device is configured by sending specific commands to it via the UART interface. It is connected to the PC with the help of a PmodUSBUART device. PC based software from Texas Instruments called Simply Blue Commander is used to send parameter setting commands to the PmodBT and set device parameters as needed. The PmodBT uses UART interface and a USB port available on the PC is used by Simply Blue Commander. A simple signal rerouting circuit is synthesized on the FPGA to easily connect the Simply Blue Commander running on a PC to the PmodBT. The PmodUSBUART acts merely as a bridge between the two interfaces.

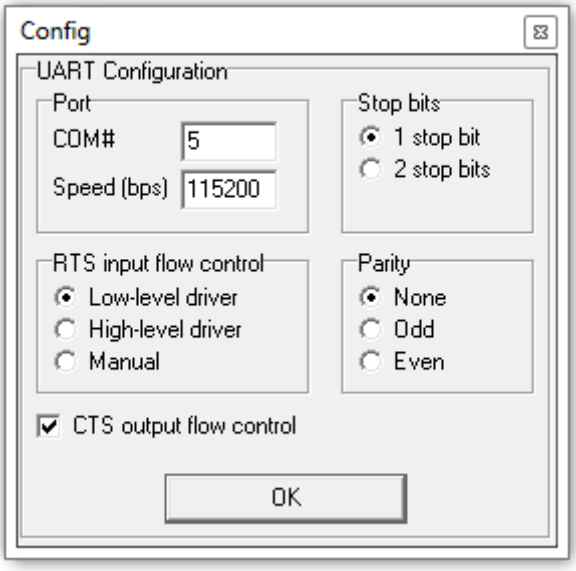

Figure II.IV. Simply Blue Commander Configuration Settings.

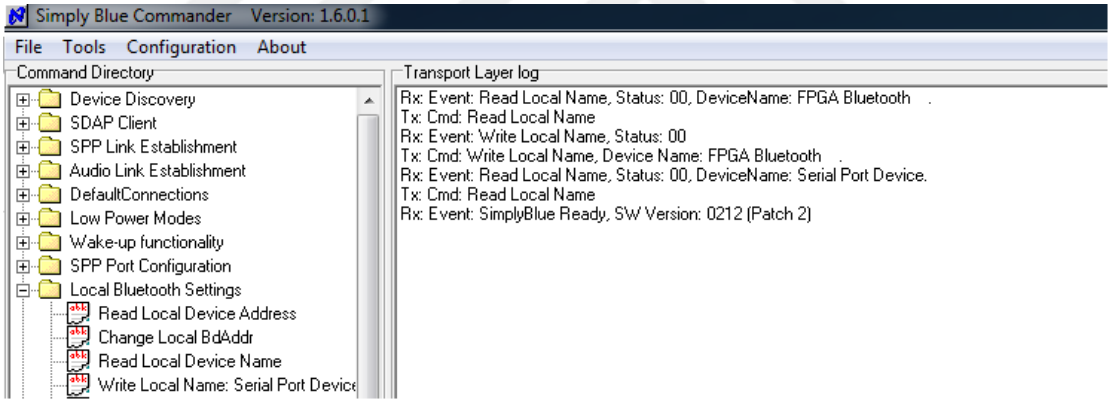

Figure II.V. Device Name Changing Commands Execution.

At first the device is set to factory defaults. Its name is changed to “FPGA Bluetooth” and its security pin is set to “1234”. Simply Blue Commander setup configurations are shown in Figure II.IV. The sending of commands and changing the PmodBt’s device name in command mode is shown in Figure II.V. As seen in Figure II.V the default device name is read, is changed as desired and is read again for conforming that the change is succesfull.

.

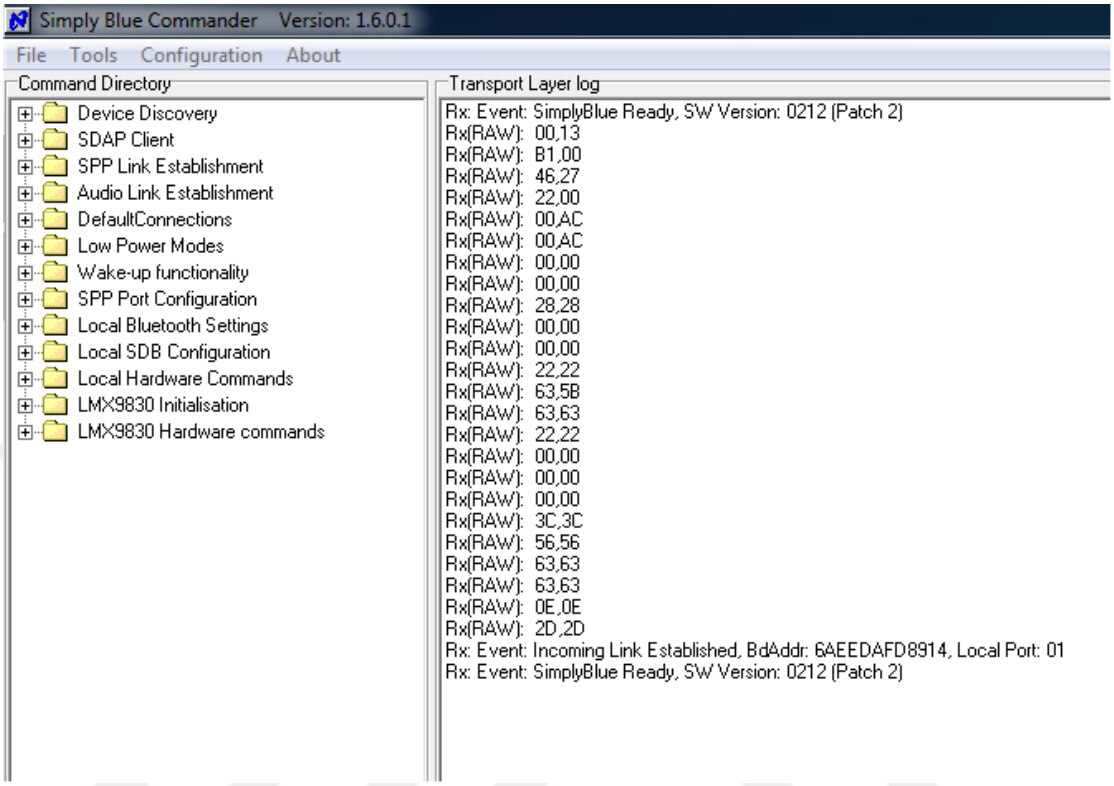

Figure II.VI. Real Time Accelerometer Readings Being Monitored in Simply Blue Commander, Sent to PmodBT From the Smart Phone.

2.10. TI LMX98938 Transparent Mode

In Transparent mode, all the data sent or received over the Bluetooth link is communicated to the UART as it is and is not translated as commands. The device by defaults is in command mode on start up, but as soon as it connects to a remote device it enters transparent mode automatically. For our system, it is assumed that the remote device will initiate a connection request with the Bluetooth device which will force the Bluetooth module to transparent mode. Hence once the basic parameters are set, there is no need for the device to be in command mode anymore. In Figure II.VI, live accelerometer data form the smart phone is being observed while the device in Transparent mode.

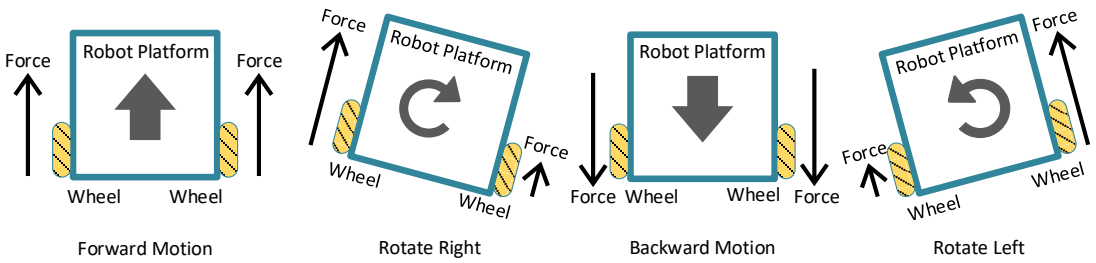

Figure II.VII. Robot Motion Behavior.

2.11. Robot Motion Behavior

There are two geared DC motors mounted at the robotic platform. The motors can reach speeds up to 300 rpm. Both the motors run independently. This implies that in order for a forward or backward motion both the motors should run at the same speed and in the same direction. A speed mismatch between the motors causes the robot to rotate.

If both the motors are moving in a clockwise direction and a speed mismatch occurs the robotic platform's rotation is always in the direction of the motor having lower speed. In case only one of the motor rotates, the rotation direction is opposite to that of the motor. For example, if the motor on the right hand side runs at a lower speed the robot will turn right i.e. clockwise. Direction is anticlockwise if true if the motor at the right hand side runs slower.

The amount of turn is proportional to the speed differences among the two motors. If the motor on the right hand side is at rest and the left hand side motor starts to spin then the robot's rotation is towards right i.e. anti-clockwise. Direction is clockwise if the motor at the left is at rest and the motor at the right hand side spins.

Similarly, if both the motors are spinning in an anticlockwise direction and a speed mismatch is present, then the robot's turn direction would be vice versa to the examples explained above. Same is true if one of the motor is at rest and the other one is spinning.

The system is designed such that the motors never run in opposite direction to each other. They run at same speed for a straight line motion and a speed mismatch is used to make the robot turn.

III. REALIZED FPGA CIRCUIT

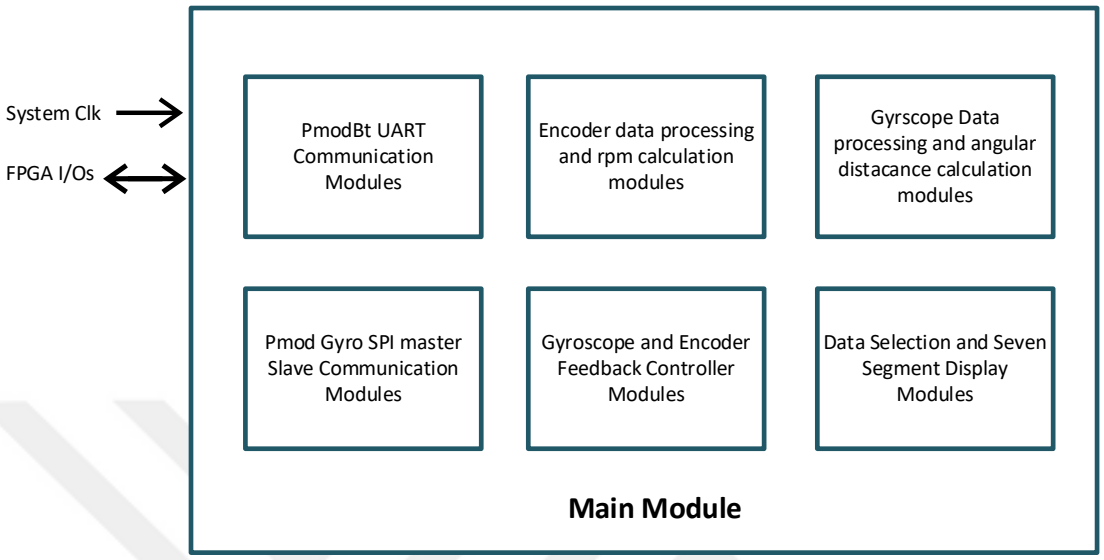

Figure III.I. FPGA Circuit Major Components Block Diagram.

3.1. FPGA Circuit introduction

The FPGA circuit consists of 25 instances of Verilog modules. The main module is set as the top module and connects the modules among themselves and with the I/O ports at the FPGA. These modules take care of the following tasks:

1) Pmod devices to FPGA and vice versa communication.
2) Bluetooth communication data selection and processing.
3) Gyroscope sensor data processing and angular position calculation.
4) Encoder data processing and rpm calculations.
5) PI closed loop control via encoder and/or gyroscope feedback.
6) PWM signal generation.
7) Display selectable data on Basys2 Seven Segment display.

The FPGA circuit close loop PI motor speed and direction controls two DC motors. It receives data from PmodBT connected to remote a device via PmodBT's

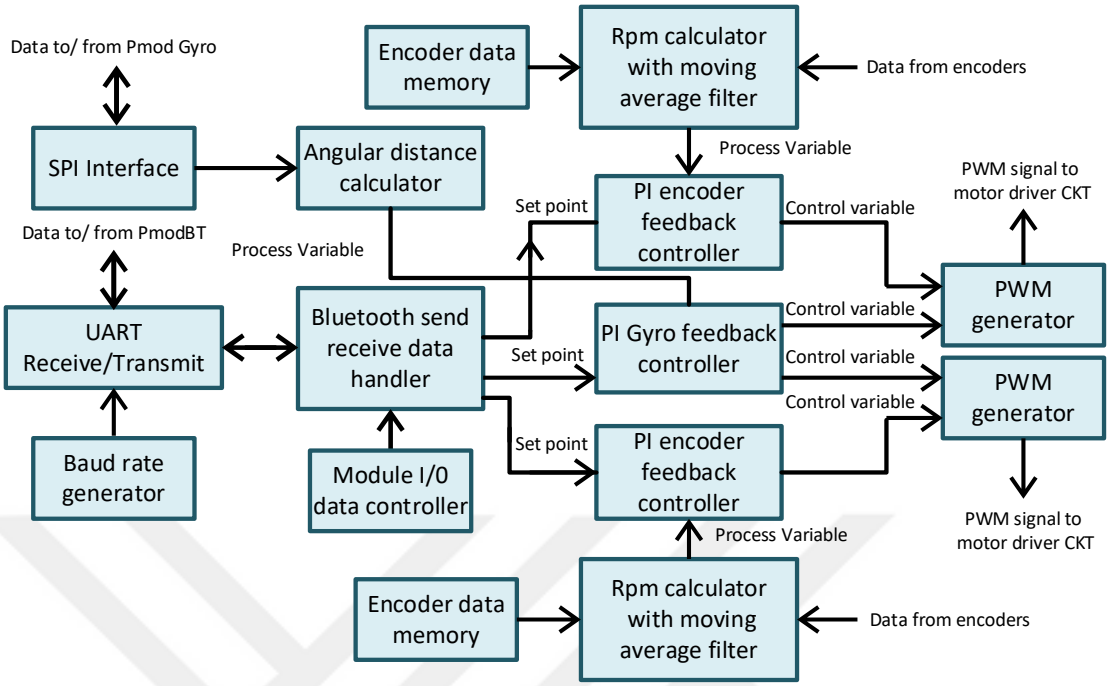

Figure III.II. FPGA Circuit Module Block Diagram.

UART interface (transparent mode). If connected to the smart phone the incoming data has motor speed (rpm) and direction information. If connected to a PC via MATLAB the incoming data can have commands and parameter settings for the circuit as well. The FPGA circuit receives pulses from rotary encoders mounted on the motor shaft. The circuit counts the number of pulses in a period of time and feeds these values to a moving average filter. The filtered no. of pulses in a set period of time is then used to calculate motor speed in rpm.

This speed data is fed to a PI controller. The controller compares the calculated rpm from encoder pulses, against the desired rpm speed set point received via the Bluetooth and adjusts PWM duty cycle accordingly.

The focus of this work is not on implementation of complex filtration and control algorithms but rather to only investigate the workability of the idea. No complex

filtration or estimation techniques are used on the sensor data and neither a complex controller is designed.

3.2. UART Communication Circuit

UART is a serial line communication interface. Our UART communication block consists of three Verilog modules. No clock information is conveyed during the UART communication, the transmitter and receiver must agree upon a set of parameters to communicate. These parameters include baud rate, number of data bits, number of stop bits and use of the parity bit. Our UART circuit uses baud rate 9600, 8 data bits, no parity bits and one stop bit. These settings are made at the PmodBT via using command mode as explained in Section 2.9 and are hard coded in the FPGA circuit.

3.2.1. Baud Rate Generator Module

Middle points of data bits received at the UART are estimated by over sampling the signals. The oversampling rate is 16 times the baud rate, which means that each serial bit is sampled 16 times.

The baud rate generator module generates an over sampled signal. The frequency of this signal is 16 times the baud rate. For our system's 115200 baud rate the sampling rate must be approximately 27 ticks per second. For our FPGA's clock rate of 50 Mhz. Parameter M in the baud rate generator module defines the required ticks per second.

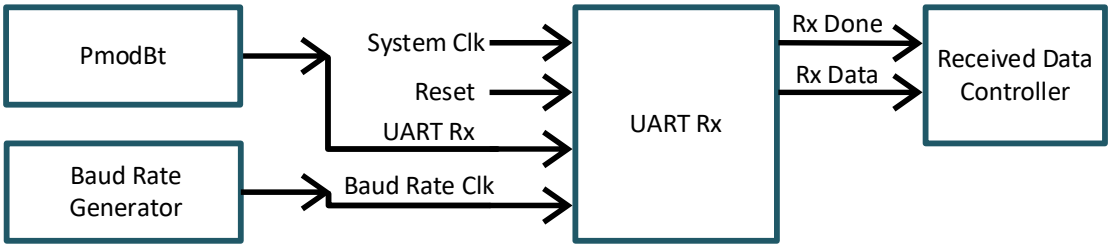

Figure III.III. UART Receive Module Block Diagram.

3.2.2. UART Receive Module

The receiver module is basically a shift register that shifts in data bit by bit and presents it as a byte. Data transmission starts with a 0 bit, followed by 8 data bits with no parity bits. A stop bit is used to indicate end of the data set.

Oversampling is used and each UART byte is sampled in the middle. The circuit waits for the 0 start bit and starts a counter for oversampling. As the over sampling rate is 16, a counter count 7 indicates the middle of the start data bit. The counter is then cleared and reset to 0 and progresses to 16. At 16 counts, the count is at middle point of the first data bit. Its value is shifted in and the counter is reset to retrieve next data bit. The steps are repeated to retrieve all 8 data bits and the stop bit.

The implemented FSM has four states namely idle, start, data and stop. Default and initial state is the idle state and an enable high is required to move it to start sate where the circuit waits for the 0 UART data start bit. At the first data bit (start bit) the circuit waits for the middle of the data bit and moves to the data state where each bit is shifted in at its middle point. After successfully retrieving the data bits the circuit proceeds to stop state where it waits for the stop bits to pass. Receive done tick is set to indicate a successful received data byte. The FSM then returns to the idle state.

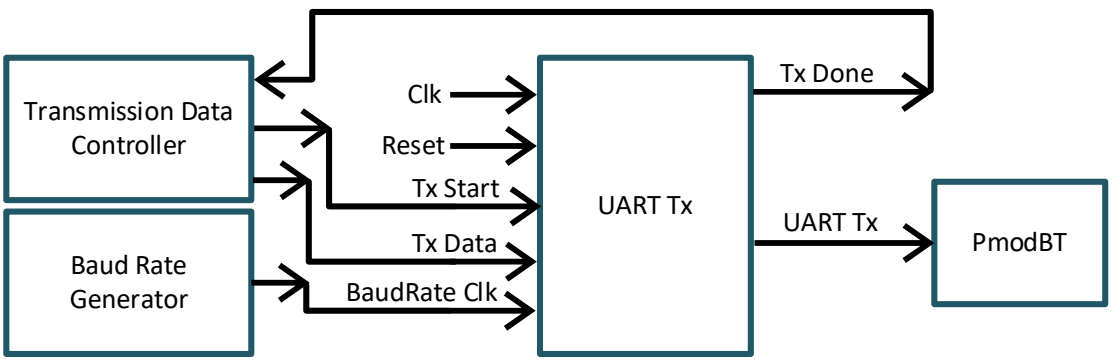

Figure III.IV. UART Transmit Module Block Diagram.

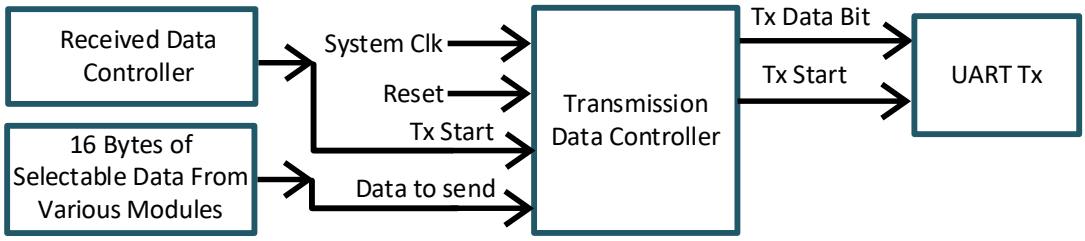

Figure III.V. Transmission Data Controller Module Connections Block Diagram.

3.2.3. UART Transmit Module

The UART Transmit module is basically a shift register that takes a byte of data and shifts it out bit by bit at a specified rate. The transmission dataset starts with a 0 start bit, followed by 8 data bits. Parity bits are not used in this system. The transmission ends with a stop bit.

The UART transmitter does not require oversampling. Hence the frequency required by this module is 16 times slower than that of the UART Receive module. It uses an internal counter to count 16 baud rate generated ticks and shifts a bit out at every 16 ticks.

The FSM implemented has four states namely idle, start, data and stop. The circuit is initially in idle state. The UART transmission line is set to high indicating no data transfer. As soon as a transmission start line is asserted the FSM moves to the start state and the transmit line is set to low indicating start of transmission. The circuit waits for 16 baud rate ticks before it moves to the data sate. At the data state 8 bits are shifted out one at a time to the transmit line. An internal counter ensures that the bits are shifted at a rate of 16 times less than the baud rate generated ticks. After shifting the eight bits,

the system moves to the stop state where the transmit line is asserted to a high and transmission done signal is asserted to indicate that the byte transmission is complete. The FSM then returns to the idle state and waits for the signal to transfer next data byte.

3.3. Bluetooth Data Handler Modules

This set of modules handle the transmitted and received data over the Bluetooth link.

3.3.1. Transmit Data Controller Module

The module handles the data that needs to be transmitted via the Bluetooth link. To observe the results and for troubleshooting a need for a system capable of retrieving FPGA circuit's internal variables in real time was felt. A MATLAB code is written which accepts data sets exactly 16 bytes wide at a time via the PC Bluetooth.

This code sends a Data transmission request to the FPGA circuit. Upon receiving the request the FPGA circuit's transmits data controller module sends 16 bytes of data with the help of UART transmit module. The variables that need to be transmitted and observed are connected at the module input ports as 16 individual bytes.

All of the FPGA circuit's sub modules are interconnected via the main module. The main module has a copy of all of these interconnecting module's variables. Any of the 16 of these variables copies can be tied to this module's input ports for transmission. As each port is only a byte wide, variables with larger bit width than a byte can be connected by using two or more input ports. Internal variables of sub modules can be observed by extracting them to the main module and putting them at the respective module's I/O port.

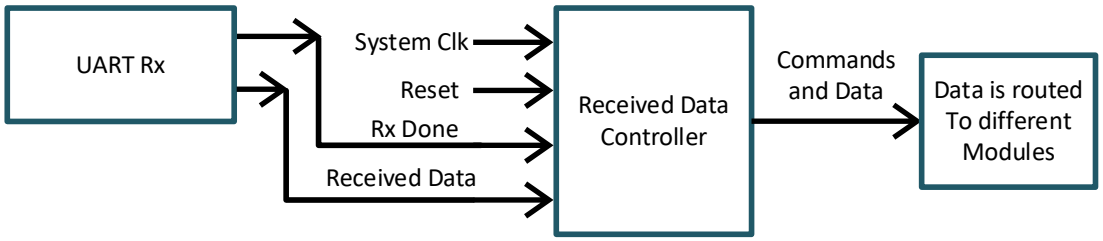

Figure III.VI. Module Received Data Controller Connections Block Diagram.

The variable set refresh rate over the Bluetooth links is limited to the UART or Bluetooth's data transfer rate. In our system, the UART communicates at a rate of 115.2 kbps, It takes 16 (no. of variables) times 10 (each variable is 8 bits plus data and a parity bit) no of bits to transfer the complete set of 16 variables. The variable data set can be refreshed at a maximum rate of 720 Hz. Observations of variables having a higher rate of change than 720 Hz is possible and an information loss is likely to occur.

The data extraction and transmission is run in parallel to other processes and does not infer a delay to other processes. The data transmission can be controlled via MATLAB or an internal enable signal. Timed data send requests from MATLAB result in extraction of data from the FPGA circuit at those time intervals. For this purpose, the MATLAB code sends data send requests every 100 ms.

The module consists of sixteen 8 bit input ports and two input bits excluding the clock and reset. The transmission start command bit controls the data set transfer. Transmission done bit is a signal form the UART transmitter module which is asserted high on successful transmission of each byte over the UART.

FSM for this module consist of three states namely, idle, select data and transmit data. The FSM waits for the transmission start command to move from idle to select data state. Each of the 16 data bytes are presented to the UART transmission module. A counter tracks the byte to be transferred. A multiplexer circuit selects the appropriate

byte with the help of the counter. Transmission start bit is asserted to high and the FSM moves to the transmit data state. It waits for the transmission complete signal from the UART transmit module. If the counter is less than 15, the counter is incremented and the next byte is transferred by moving the FSM to select data state. If not the FSM moves to idle state and waits for the next transmission start command.

3.3.2. Received Data Controller Module

This module processes the incoming Bluetooth data. The incoming Bluetooth data has various commands and parameters for the FPGA circuit. Each command or a parameter value is preceded by a specific code. This module retrieves this information and routes them to a module where it is needed.

This module apart from the clock and reset signals has a received done tick and an 8 bit wide received data port. Received done tick indicates successful read of a byte by the UART receive module and the received data port has the received UART data. Output ports of different sizes route received information to other modules. The codes and the data they represent and modules which utilize them are briefly summarized in Table III.1 and Table III.2.

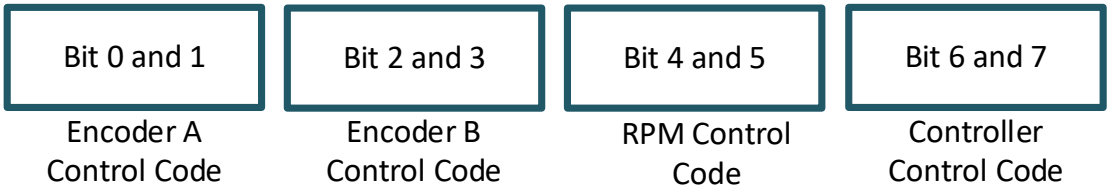

Figure III.VII. Control Code Byte Individual Commands.

Table III.1. System Commands and Control Codes

Codes	Module	Description
61680	UART transmission	Start UART transmission
00513	PI controller encoder feedback motor a	Set point for motor a rpm
01027	PI controller encoder feedback motor b	Set point for motor b rpm
01541	Multiple modules see Table III.2	Multiple uses see Table III.2
02055	MATLAB control	Pseudo motor a rpm for simulation
02569	MATLAB control	Pseudo motor b rpm for simulation
03083	PI controller encoder feedback motor a PI controller encoder feedback motor b	P gain parameter value for encoder feedback based controllers
03597	PI controller encoder feedback motor a PI controller encoder feedback motor b	I gain parameter value for encoder feedback based controllers
04111	PI Gyroscope feedback controller	P gain parameter value for Gyroscope feedback based controller
04652	PI Gyroscope feedback controller	I gain parameter value for Gyroscope feedback based controller

The FSM for this module has only two states namely idle and select data. The module uses a data shift register which accommodates three bytes of data. On successful UART byte read, the byte it is shifted in and the oldest one is discarded. The first 16 bits (2 bytes) from MSB are data code and the third byte is the data byte. On a successful read of a byte by the UART the FSM shifts in the byte and moves to the select data state. As each byte of information is preceded by two bytes of code therefore three bytes are required for a byte of data to be routed to its desired destination. This comparison occurs at every byte transfer and if a match is not found for the preceding two bytes of code the data is discarded.

Table III.2. Received Data Code Commands for MATLAB Control

Bits	Value	Description
0-1/2-3	00	Set motor encoder A/B value to value obtained from encoder pulse processor module.
	01	Overwrite motor encoder A/B value to high.
	10	Overwrite motor encoder A/B value to low.
	11	Set motor encoder A/B value to a value obtained from encoder pulse processor module (default).
4/5	0	Set RPM value for control modules to calculated RPM from RPM calculated module (default).
	1	Set RPM value for control modules to received RPM value from MATLAB
6	0	Stop gyro feedback based robot's angular position controller.
	1	Start gyro feedback based robot's angular position controller.
7	0	Stop encoder feedback based robot's motor A/B wheel speed controller.
	1	Start encoder feedback based robot's motor A/B wheel speed controller.

3.4. MATLAB Command Control Module

The MATLAB command control module manipulates calculated motor rpm and encoder values in accordance with the information received from MATLAB. Table III.2 summarizes the byte values and their operation performed by the module. Encoder and calculated rpm values pass through this module before they are used anywhere else in the system. In case of an encoder failure the encoder pulse may remain at a high or a low. In either case the rpm calculated is always 0. Disconnecting the encoders implies that the rpm values can no longer be calculated by the FPGA circuit.

The encoder High and low ticks are processed by "Encoder Pulse Processor" module. This module is connected to the encoder pulse line from the pmodHB5. A high or low pulse is only recognized only if the encoder pulse remains high or low for 8 consecutive clock cycles. This effectively filters out glitches in encoder pulses.

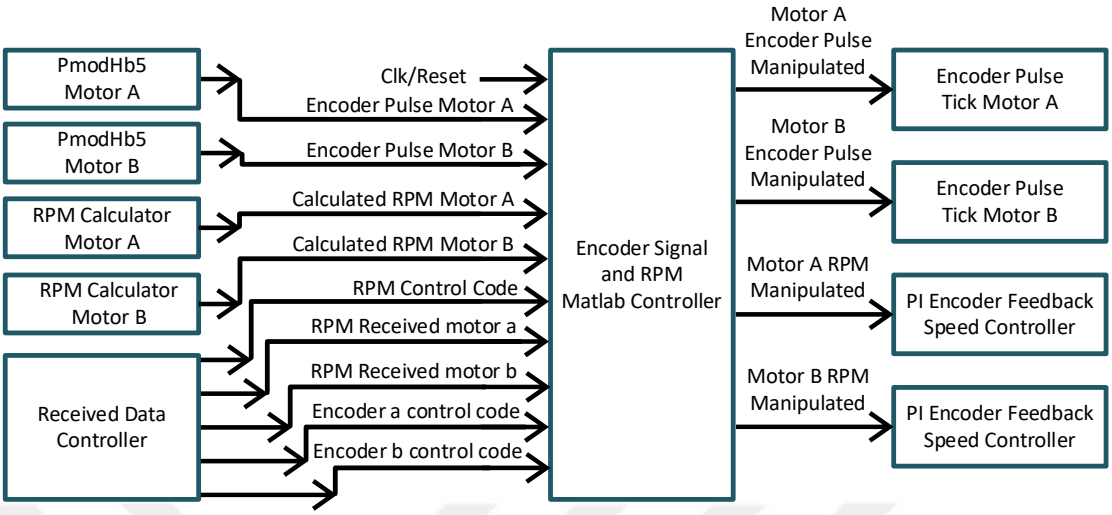

Figure III.VIII. Module Encoder Signal and RPM MATLAB Controller Connections Block Diagram.

The same module also has the functionality to measure each pulse's Time Period and pulse counts per revolution and total pulse counts. Initially this module is used to obtain the ppr of the encoder by manually rotating the wheel exactly one revolution. Once the encoder ppr information is obtained, there is no need to re-obtain it in the system.

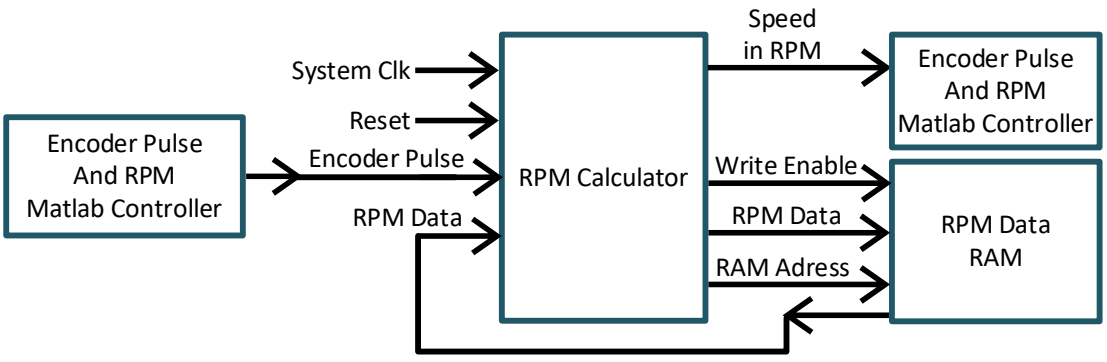

Figure III.IX. Module RPM Calculator Connections Block Diagram.

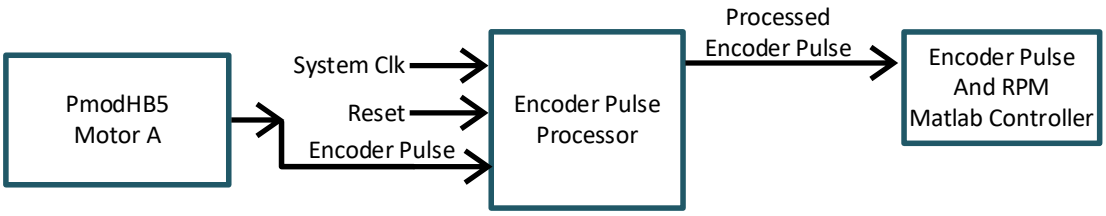

Figure III.X. Module Encoder Pulse Processor Connections Block Diagram.

The Pulse Time period information is also not used. Both the functionalities are present in the module but as they are not used in the system, their connections are also omitted from the figure.

3.5. RPM Calculator and RPM Data RAM Modules

Encoder pulse count method is used for speed estimation. The encoder's resolution is 58 ppr. Hence the rpm can be calculated by using the following mathematical expression.

$$Speed\ (rpm) = (\frac{EncoderPulse\ Count\ 60}{Second})/58 \qquad \text{(III.1)}$$

where Encoder pulses / revolution = 58

The module counts the number of pulses in 1.034 seconds to obtain motor speed in rpm. The module continuously feeds no. of pulses in 1.034 seconds to a moving average filter. This filter uses a 6 bit wide 32 slot memory RAM. On every 32.32 ms a fresh value (encoder pulse count) is written to the memory with incrementally overwriting all memory the addresses, an average of these values is calculated. The average is the motor rpm speed with noise suppressed. This speed is fed to the encoder feedback based PI motor controller module as the controller's process variable.

In order to conserve FPGA resources, we try to eliminate as many mathematical calculations as possible. An analogy has been adopted in this module. The following

equation gives us motor rotation speed in rpm without performing any division or multiplication operations at all.

$$Speed\ (rpm) = \left(\frac{EncoderPulses}{1.034}\right). \quad \text{(III.2)}$$

For wheel rpm, the module only counts the no. of encoder pulses occurring in 1.034 seconds which is the same is wheel rpm.

3.6. Module PI Speed Control

This module close loop speed controls motor rpm with encoder feedback. The PI gains are adjustable via MATLAB. Two instances of this module are implemented, one for motor “A” and the other for motor “B”.

The FSM has three states namely delay, calculate error and calculate output. The default state is the delay state and sets the rate of control variable calculation which is 47.5 Hz. The calculate error state calculates the error and output control variable value. Error and the control variable are calculated based on the difference between the set point and process value (or vice versa if the process value is larger). If the calculated value of the variable that holds the control variable’s value exceeds its maximum value that is 2^{16} the variable overflows.

To prevent this from happening the variable is set to maximum value and remains there unless the control variable’s value decreased by the calculate error state.

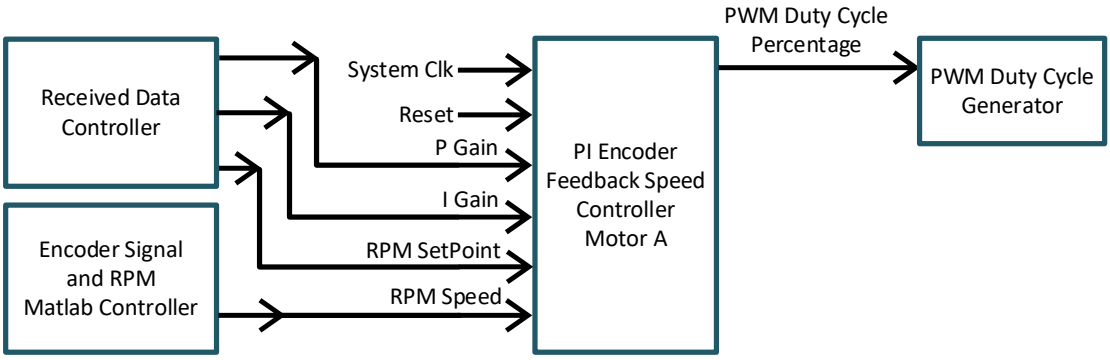

Figure III.XI. Module PI Encoder Feedback Speed Controller Motor A Connections Block Diagram.

3.7. SPI Interface Module

The PmodGyro houses a L3G4200D STMicroelectronics. This device can utilize two communication interfaces I2C and SPI. In this system only SPI is used. The L3G4200D can measure angular rate of change in three axes. Each axis value is 16 bits wide. Angular rate of change scale value is selectable from 250, 500 and 2000 dps. Our system uses 250 dps angular rate of change which gives the highest precision. Each digit in this setting is 8.75 mdps.

The device has a 32 slot 16 bit FIFO buffer for each of three Yaw, pitch and roll. This FIFO buffer can be used in five different modes. In this system, the buffer is used in its default bypass mode. Only the first slot 16 bit slots are used and on the arrival of new data the old data is overwritten.

SPI mode three is used i.e. the SDI and SDO are driven at the falling edge of SPC and are captured at the rising edge of SPC.

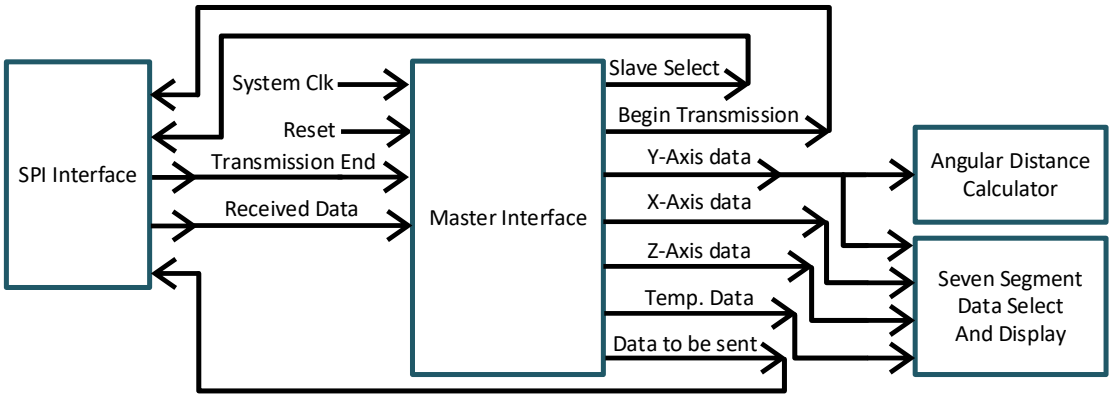

Figure III.XII. Module Master Interface Connections Block Diagram.

A set of registers are used to control the device's behavior and to retrieve its data. These registers can be accessed via their 6 bit addresses. The 6 address bits are preceded by a read/write and an increment select/deselect bit. If the read/write bit is 1 data at the address is read, if 0 data is written to the address. The data to be written and read is followed by the address data set at MOSI-SDI and MISO –SDO line.

The SPI interface module's input ports consist of 8 bit send data port, a begin transmission bit, a MISO bit, clock and reset bits. Output ports consist of 8 bit received data port, an end transmission bit a MOSI bit and a synchronous clock. The FSM consists of three states namely idle, receive transmit and hold.

The FSM is initially in idle state. If a begin transmission bit is asserted high the FSM moves to receive transmit state where a byte of data is shifted out and in bit by bit via MOSI-SDI and MISO-SDO lines. This data receive and transmission is done in accordance with SPI mode3. After the operation is complete and end transmission signal is asserted to indicate a successful receive/ transmission of a byte to master interface module. The FSM moves to hold state. An asserted slave select by the master

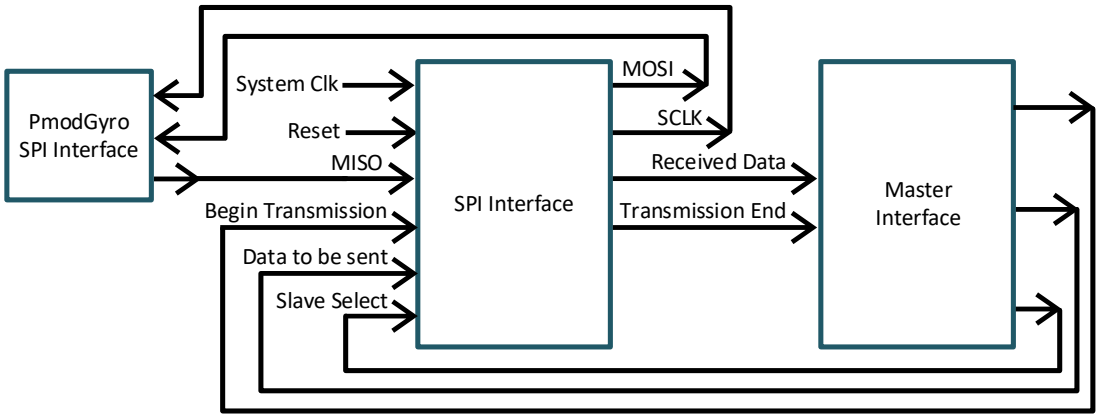

Figure III.XIII. Module SPI Interface Connections Block Diagram.

interface module implies that there is no more data to be transmitted/ received. In this case, the FSM is routed back to idle state. If a begin transmission bit is asserted the FSM is routed to receive transmit state. A separate always blocks generates SCLK signals at a rate of 12.2 kHz. This clock is generated only when receive or transmit operation is in progress.

The master interface module controls the data that needs to be transmitted from and to the PmodGyro via the SPI interface. In order to use L3G4200D some parameters need to be set. In this system, the "CTRL_REG1" needs to be set for selecting Output data rate, bandwidth and enabling the Z, Y and X axis gyroscopes. All the other control variables are set to default values and there is no need to reset them. Once the control variables are set the FPGA circuit only needs to send periodic data read commands to the device and retrieve the incoming data.

The FSM consists of 7 states, namely idle, setup, temperature, run, hold, wait slave select and wait run. Default state is the idle state. On a start assert the FSM moves to setup command. If not done the control register is initialized. The control value 0Fh which sets the data output rate to 100 Hz and enables gyroscope in X, Y and Z axis

preceded by 20h (address of the control register) is sent to the device one byte at a time. FSM waits for the byte transfer in hold state and returns back to setup state on transfer complete. Once the initial setup is completed, the FSM moves to the slave select and wait run sate progressively.

The delay between each data read is adjusted in wait run state. The states run and temperature send commands to retrieve temperature and gyroscope data. The decision for next state is made upon the previous state information. Such that if the temperature data is read previously via temperature state the next data to be read is gyroscope registers via run state and vice versa. The default is temperature state. The command for reading temperature data is 26h but the command code is preceded by a high read and low increment address bit. Code A6h is sent to the device which implies that the temperature data is read from the OUT_TEMP register. The FSM moves to hold sate waits for the byte transfer to complete and sends the FSM back to temperature state. Once the command is sent to the device the next to be received byte has the temperature data hence a 00h is sent to the device and temperature data is retrieved in hold state. The FSM proceeds to wait state, as the temperature data has been retrieved the wait state sends the FSM to run state.

The run state a sends command for retrieving X, Y and Z axis gyroscope data. The address for reading “OUT_X_L” register that has the lower significant byte of the x axis gyroscope data is 28h. The other data sets are in a sequence at incremented addresses from the “OUT_X_L” register, for e.g. “OUT_X_H” is at 29h “OUT_Y_L” is at 30h. E8h is written to the device as the data read bit and address increment bits are asserted. If the address increment bit is asserted the read address is automatically

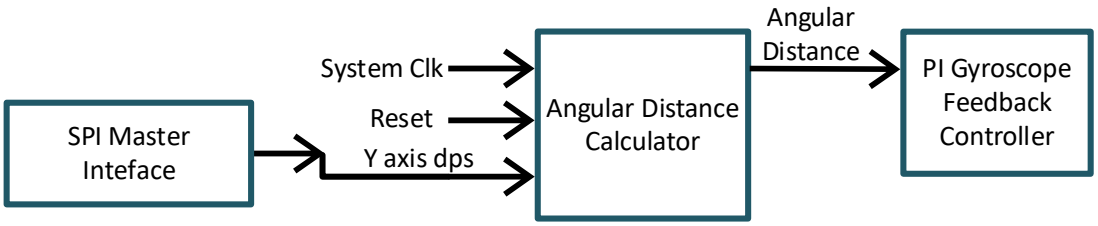

Figure III.XIV. Module Angular Distance Calculator Connections Block Diagram.

incremented in multiple read commands. The FSM moves to hold state and waits for the byte to be transferred and returns to the run state.

The address is incremented to retrieve next register's data. The FSM retrieves data from all 6 registers by incrementing data address, moving to the hold state waiting for the byte transfer, collecting data and returning to the run state for 5 more times. The FSM returns to the wait run state and if a start is not asserted to 0, the FSM infers a set delay and proceeds to read temperature data again.

3.8. Distance Calculator

The distance calculator module integrates the angular rate of change in Z-axis (Yaw axis) of the gyroscope. The MEMS gyroscopes exhibit a steady state drift. To prevent this steady state, drift the values are integrated only when the robot is in motion. An observation of the steady state drifts with the robot in a stationary state is done. Any angular rate of change higher than these observations in any of the axis implies that the robot is in motion. The values are 4.5 dps for X axis and 2.25 dps for Y axis.

The last two bits of the rate of change are truncated. This induces quantization error but is beneficial in slicing of gyroscope drift error. The Z axis angular speed is integrated at a rate of 256 Hz. The FSM for this module consists of only two states

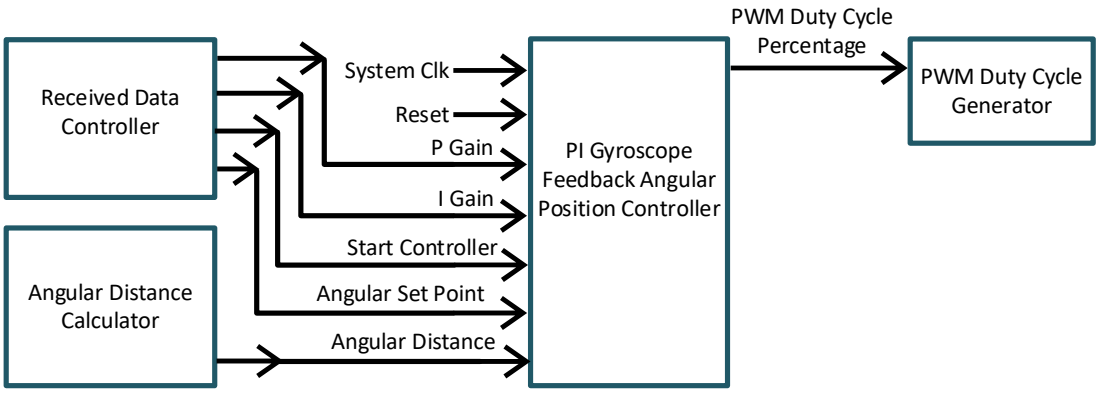

Figure III.XV. Module PI Gyroscope Feedback Angular Position Controller Connections Block Diagram.

namely delay and integrate. The default state is the delay state. The FSM waits for a specific time period in the delay state and if the condition of robot motion is found to be true the FSM moves to the integrate state.

The default state is the delay state. The FSM waits for a specific time period in the delay state and if the condition of robot motion is found to be true the FSM moves to the integrate state. The gyroscope value is integrated and a 2's complement based subtraction operation is performed if the angular speed is found to be negative.

3.9. Gyroscope Feedback Based Angular Position Controller

In case of a total encoder failure calculated rpm becomes 0 as no pulse ticks are registered. Similar is the case for encoder pulse stuck at a low or a high. The encoder feedback based controller by design progressively starts to increase PWM duty cycle which increases motor speed. This results in a speed mismatch between the speeds of the two motors and the robot starts to deviate from its path. This angular deviation is continuously calculated and monitored from the gyroscope data and is fed to the Gyroscope feedback based PI controller which attempts to correct the error in PWM

duty cycle. The angular distance values range from 0 to 8129 for 0 to 360 degrees for robot's anticlockwise rotation.

This module PI regulates the robot's angular orientation in Yaw axis using angular distance covered in Yaw axis as feedback. This angular distance is calculated form angular rate of change obtained by an onboard gyroscope. The angular deviation is corrected via manipulating speed of only one of the motors. The PI gains of the module are adjustable via MATLAB. The controller is applied only on one of the two motors at a time. In this system, it is applied to the motor that is the motor on the left hand side named motor A. An analogy is developed to be used in this module.

For the robot to move forward both the motors spin in clockwise direction. A wheel spin speed mismatch causes the robot to turn. The robot will move clock wise in Yaw axis in a condition where the left hand side motor spins at a greater speed the right hand side's motor and anticlockwise if the left hand side motor's spinning speed is less than the right hand side motor's spinning speed. The robot turns clockwise when the motor A's speed increases and turns antilock wise when the speed decreases. The PI controller module increases the motor's speed to turn the robot clockwise or correcting anti clock wise deviation and reduces the speed to turn anti clockwise and to correct clockwise deviation. The robot must also follow the shortest path for deviation correction or turning to a set point.

The process variable is the distance covered calculated from the "angular distance calculator" module whose value is 13 bits wide. The set point is the desired angular direction and is adjustable via MATLAB. The output is a 16 bit wide control variable which is fed to PWM generation module for the motor.

For correct speed adjustment and to make the robot always follow the shortest path an analogy has been used between the MSB of the angular position variable and the MSB of the error variable. If the set point is less than the process variable and both the MSB's of the angular position and error variables are same i.e. 1 1 or 0 0 the wheel speed is increased and decreased if they differ. The opposite is true if the set point is larger than the process variable. If the MSB's are same the speed is decreased and increased if they differ.

For example, if the current angular position is 60 degrees and the set point is 30 degrees, both the MSB's are same and the speed is increased to turn the robot clockwise from 60 to 30 degrees. On the contrary if the set point is 60 degrees and the robot is at 30 degrees the speed is decreased to turn it anti clock wise. Please refer to figure for examples on possible scenarios.

The PI gains are adjustable via MATLAB. The FSM has three states namely delay, calculate error and calculate output. The default state is the delay state and sets the rate of control variable calculation which is 47.5 Hz. The calculated error state calculates the error and output control variable value. Error and the control variable is calculated based on the difference between the set point and process value (or vice versa if the process value is larger). If the calculated value of the variable that holds the control variable's value exceeds its maximum value that is 2^{16} the variable overflows. To prevent this from happening the variable is set to maximum value and remains there unless the control variable's value decreased by the calculate error state.

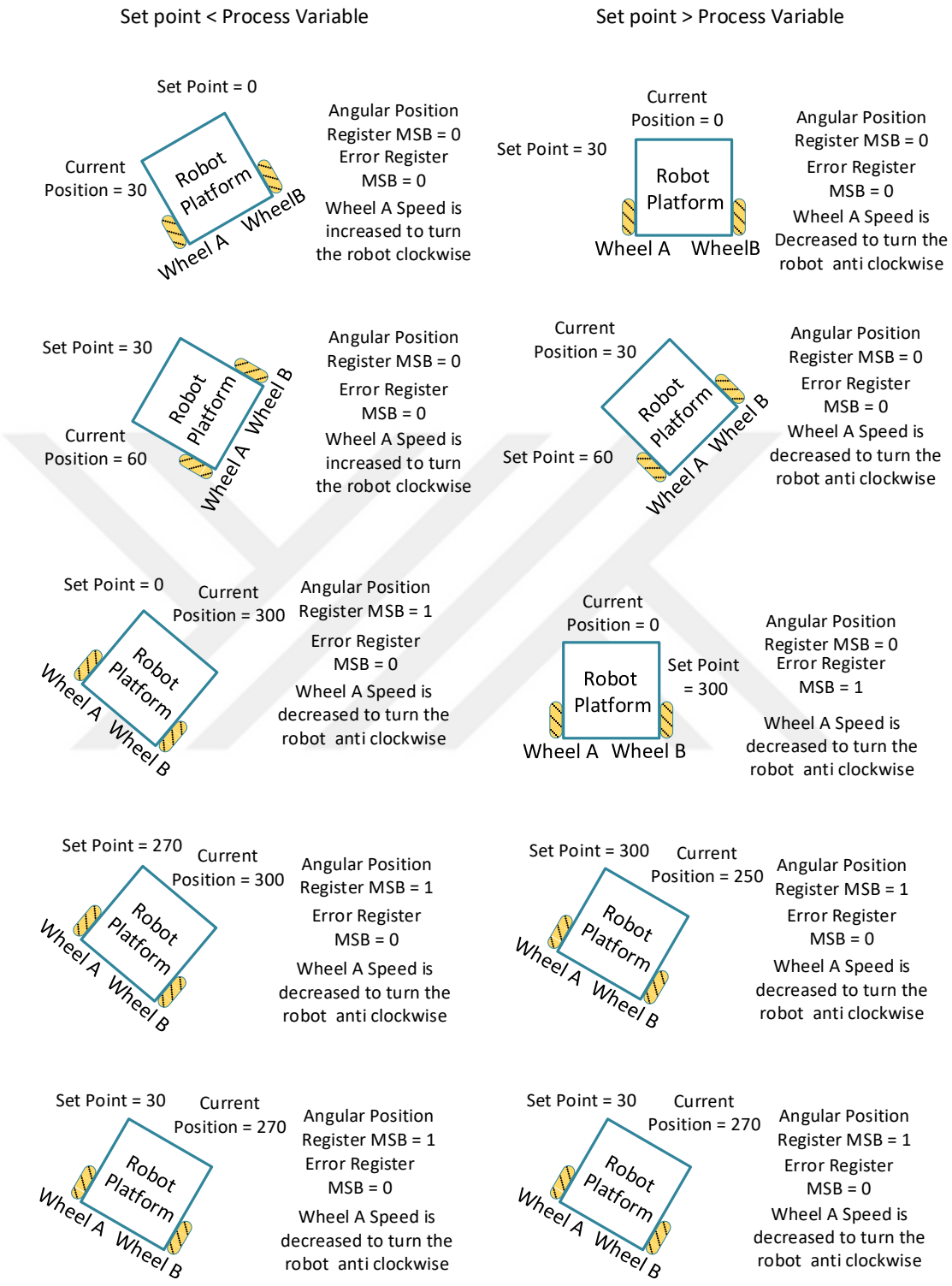

Figure III.XVI. Direction Decision Making Analogy Block Diagram.

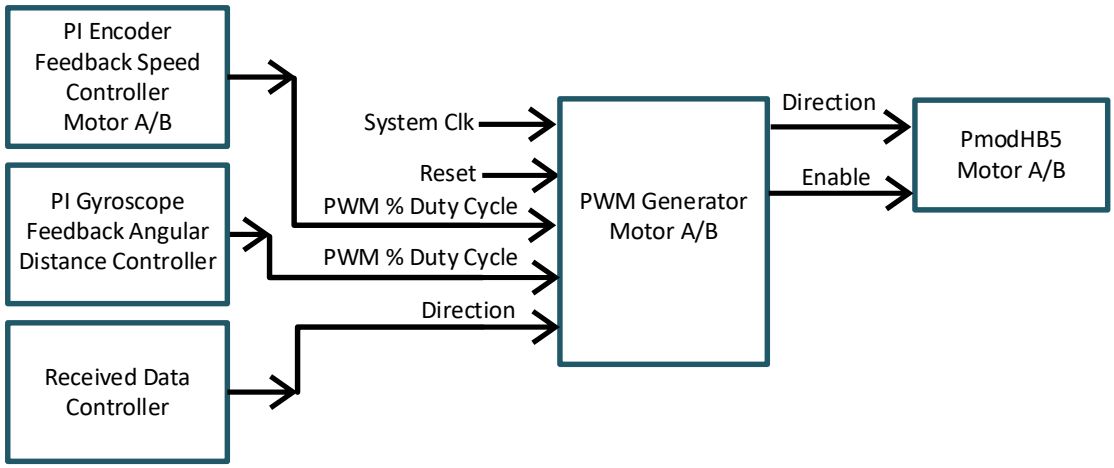

Figure III.XVII. Module PWM Generator Motor A/B Connections Block Diagram.

3.10. PWM Generation Module

PWM scheme is used to control the motor's speed. This module generates PWM duty cycles and direction control signal for DC motor speed control. The control variable from the controller module represents 0.0122% duty cycle per digit. The PWM signals have a frequency of 1.525 kHz. The pmodHb5 employs an H bridge circuit to control the direction of the motor. An instantaneous motor direction switching can damage the H bridge circuit because the transistors in the circuit need a very small but a finite period of time to turn on or off. This induces a short circuit for a short instance of time.

In order to prevent damage to the H bridge circuit, before every direction change the enable pin is set to a low for 150 ms, the direction pin is changed and again a delay of 150 ms is inferred before setting the enable pin to high.

The FSM for this module consists of only three states, delay, enable high and enable low. Default state is the delay sate. The direction switching is taken care of in this state. If there is no direction switch this sate infers no delay and the FSM transits

to the enable high state. The enable signal is set to a high for a time defined by the control variables and the FSM transits to enable low state. The enable pin is set to a low for the reaming time period of the PWM signal and the FSM transits to the delay state again.

3.11. Data Display Modules

A set of four seven-segment Led display on the FPGA kit is present on the Basys2 Kit. A set of three modules display real time data on the FPGA kit's on this display. The data that needs to be displayed is selected via switches "N3, E2 and F3" on the FPGA kit, Table III.3 summarizes the switch states and the data displayed. The data switching task is handled by the data select module. The data is converted from Hex to BCD by "Hex to BCD" module.

The "seven segment display" module displays data on the display. To save resources on the FPGA kit the seven-segment displays share the eight led light pins and each one has individual enable pin. The module generates time multiplexed signals to turn each display on one at a time and presents BCD to seven segment decoded data at its led pins the very instant to the display. All the values that are greater than 12 bits are truncated to 12 bits.

Table III.3. Switch Selections

Switch N3	Switch E2	Switch F3	Description
0	0	0	X axis angular rate of change
0	0	1	Y axis angular rate of change
0	1	0	Z axis angular rate of change
0	1	1	Temperature
1	0	0	Motor A rpm
1	0	1	Motor B rpm
1	1	0	Angular distance covered
1	1	1	Angular distance covered

IV. EXPERIMENTAL RESULTS

4.1. Test Conditions

The Experiments are performed at the “Power Electronics and Energy Conversion Laboratory”, Faculty of Engineering and Architecture, Okan University, Tuzla Campus. The robot is run on a slightly rough and flat ceramic tiled surface. The MATLAB code is used to generate test stimuli and observation of the results for the Experiments.

Experimental tests are performed to assess the performance of the system. All the experiments are performed on only one of the two motors with the other one running normally and hence its data is also not presented. Controller parameters for encoder feedback based controller are set as $k_p = 64$ and $k_i = 8$. For gyroscope feedback based controller $k_p = 256$, $k_i = 8$.

4.2. Results and Observations

In all the experimental result figures, top graphs present the two controller’s control variables. These control variables are thirteen bits wide and have 0.0122% duty cycle value per digit. Black line plots the Encoder feedback controller control variable. This variable is overflow protected. Gray line plot is Gyroscope feedback controller control variable which is allowed to overflow by design and is the reason behind the flipping of variable values from minimum to maximum and vice versa. This overflow allows subtraction operation based on 2’s complemented addition without the need of

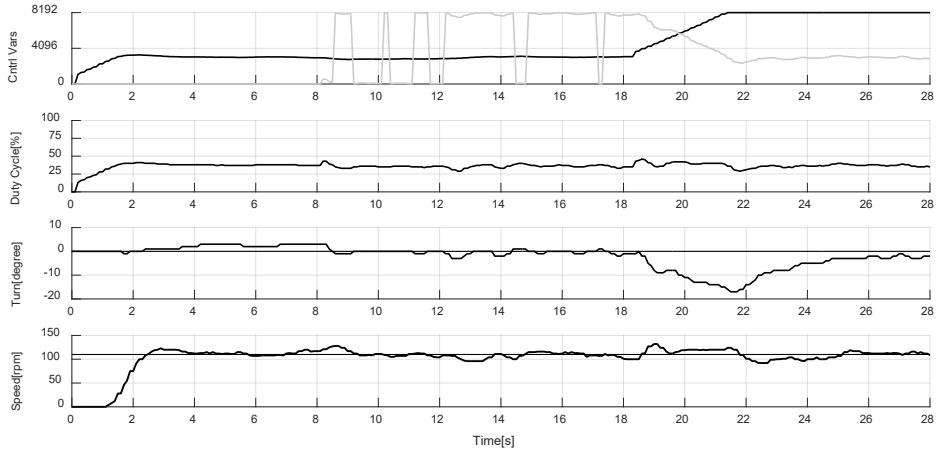

Figure IV.I. Experimental Results with Encoders in Normal Mode.

a separate subtraction circuit. Second graph in each circuit present the % duty cycle obtained from the resultant of the two control variables.

Third graph presents robot's angular deviation (Yaw axis) in degrees. Fourth graph presents motor's speed in rpm.

4.2.1. Encoders Running in Normal Mode

In **Error! Reference source not found.**, initially both the encoders run as normal and the system is allowed to gain momentum. An approx. 5 degrees deviation is gained till 8 s. At this instant, the gyroscope feedback based controller starts and rectifies the deviation. At 18 s, the motor encoder is disconnected. From this instant, the encoder feedback based control variable progressively proceeds to maximum value and the gyroscope feedback based control variable attempts to counter balance it. From 18 s to 22 s, a 20 degrees clockwise deviation is recorded which rectifies between 22 to 24 s. A small raise and loss in rpm is also recorder at these instants.

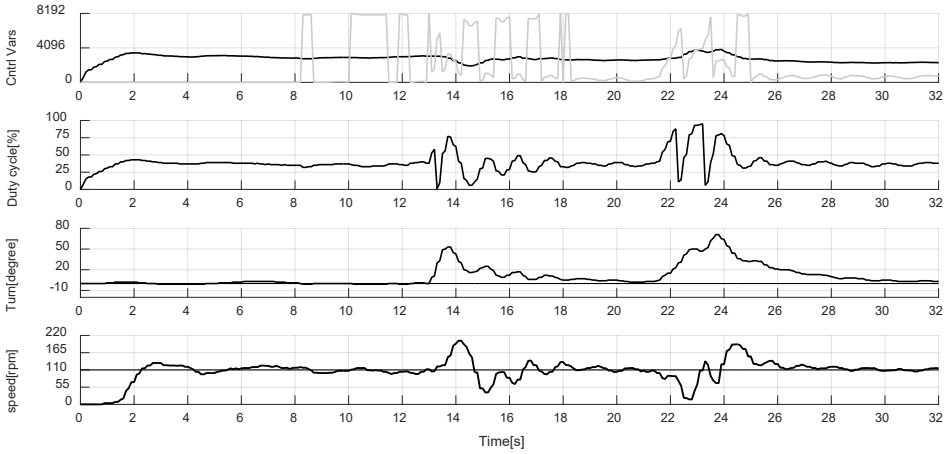

Figure IV.II. Experiment Results with Encoder Running Normal and External Disturbances.

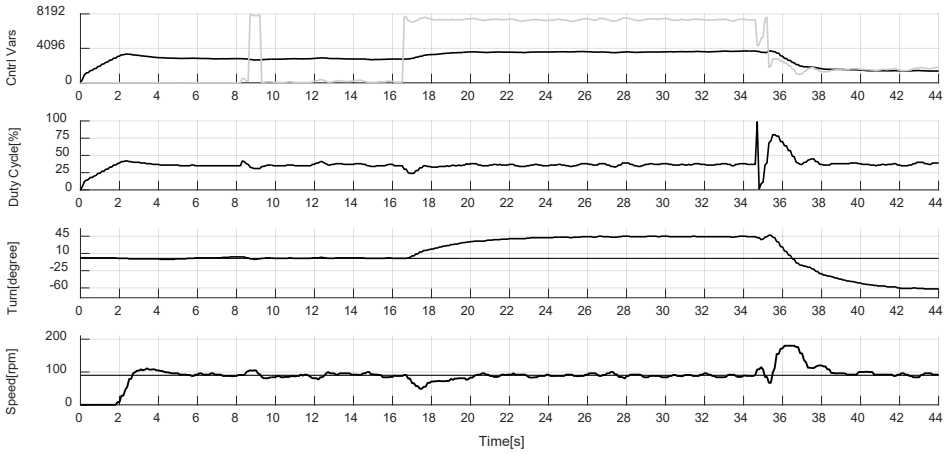

Figure IV.III. Results with Encoders Running Normally and Changing Set Points.

4.2.2. Encoder Running Normally and the Robot is Subjected to External Disturbances

In Figure IV.II, both the encoders run normally through the test. The robot is subjected to physical disturbances at 13 s and 21 s. At 13 s, a wheel slip takes place which raises the wheel speed. At 23 s, an obstacle lessens the wheel speed. As seen in

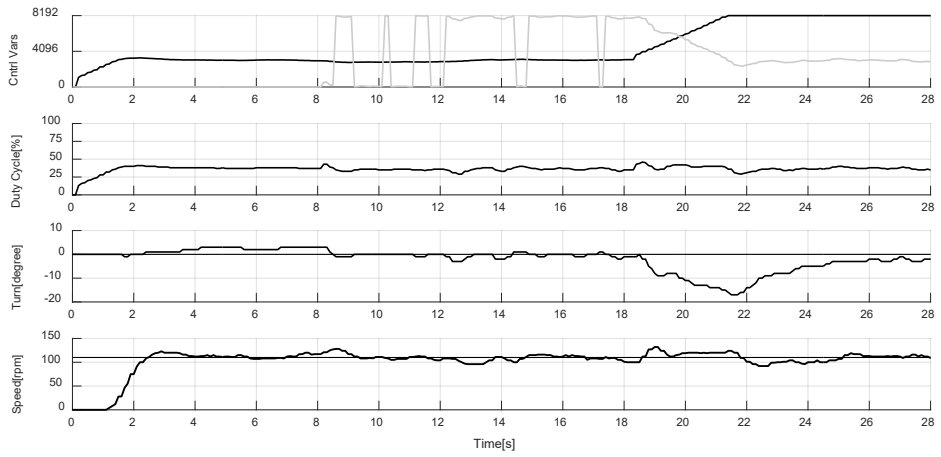

Figure IV.IV. Results with Encoders Disconnected and Varying Angular Set Point.

the figure this disturbance results in an angular rotation of the robot which is corrected by the Gyro controller.

4.2.3. Encoders Running Normally and Changing Angular Set Point

In Figure IV.III, both the encoders run normally. The direction set point is set at 45 degrees and -60 degrees at 16 and 36 respectively. Both controllers run normally, when the gyroscope feedback based controller attempts to alter wheel speed the encoder feedback based controller recognizes it as an error and tires to correct it. Due to this a small rise at 17 s and a decline at 36 s in its value is observed. As the PI gains for the gyroscope feedback based controller are much higher the robot is easily directed to the set angle by it.

4.2.4. Results with Encoders Disconnected and Varying Angular Set Point

In Figure IV.IV, the system is initially allowed to gain momentum and the encoder is disconnected at 9 s. The direction set point is reset to 45 and -60 at 19 s and 35 s respectively. An overflow in control value results in a spike at 35 s value. The robot turns to the specified angles with only gyroscope feedback based closed loop control.

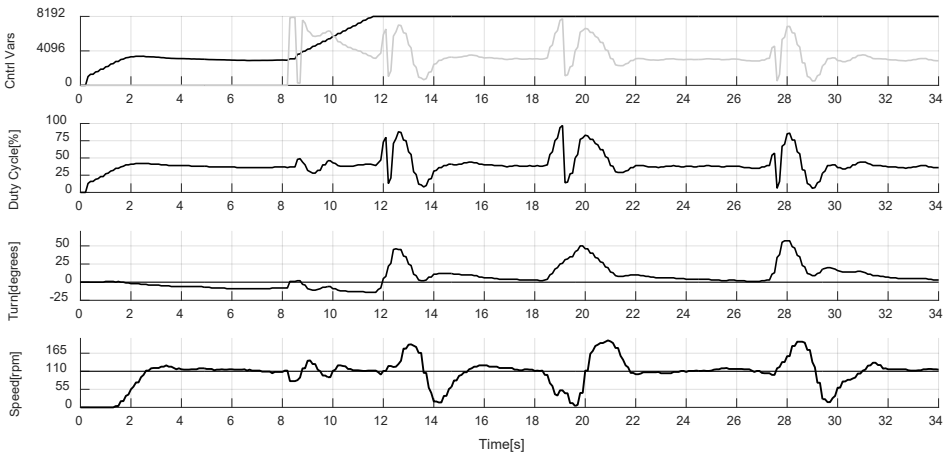

Figure IV.V. Encoders Disconnected and External Distortions.

4.2.5. Encoders Disconnected and External Distortions

In Figure IV.V, the encoders are disconnected at 8 s. The system is subjected to distortions at 13 s, 18 s and 27 s. As seen in Figure IV.V, the robot returns to its set point successfully with encoder in the failed state and using only gyroscope based closed loop controller.

V. CONCLUSION

A dual sensor feedback motor control algorithm for a robotic platform is proposed. The purpose of this research is to demonstrate that inertial sensors can improve robustness in the systems like discussed in this document. Our practical setup successfully demonstrated ability to resist direction deviation and maintain functionality in case of an encoder failure with the help of a gyroscope sensor.

Similar systems already employing encoders and gyroscopes can use this method for increasing fault tolerance and control robustness without any added hardware. Future works may include implementation of more complex filtration and control algorithms to the current system.

:

REFERENCES

[1] F. Aldhaban, “Exploring the Adoption of Smartphone Technology : Literature Review,” *Picmet ’12*, pp. 2758–2770, 2012.

[2] V. Guimarães, D. Ribeiro, L. Rosado, and I. Sousa, “A smartphone-based fall risk assessment tool: Testing Ankle Flexibility, Gait and Voluntary Stepping,” *IEEE MeMeA 2014 - IEEE Int. Symp. Med. Meas. Appl. Proc.*, 2014.

[3] X. Li *et al.*, “Smartphone evolution and reuse: Establishing a more sustainable model,” *Proc. Int. Conf. Parallel Process. Work.*, pp. 476–484, 2010.

[4] P. N. A. Fahmi, P. Budhi, J. Song, Ardiansyah, D. Choi, and Y. Kim, “3D-to-2D projection algorithm for remote control using smartphone: Enhancing smartphone capability for costless wireless audio visual consumer appliance control,” *Proc. - 27th Int. Conf. Adv. Inf. Netw. Appl. Work. WAINA 2013*, pp. 1044–1049, 2013.

[5] M. M. Makki, G. A. Saade, A. G. Altouma, S. Al-Terkawi, A. Baobeid, and R. Tafreshi, “Acquiring and analyzing electrocardiograms via smartphone to detect cardiovascular abnormalities,” *2014 IEEE-EMBS Int. Conf. Biomed. Heal. Informatics, BHI 2014*, pp. 277–280, 2014.

[6] L. Cantelli, G. Muscato, M. Nunnari, and D. Spina, “A Joint-Angle Estimation Method for Industrial Manipulators Using Inertial Sensors,” *IEEE/ASME Trans. Mechatronics*, vol. 20, no. 5, pp. 2486–2495, 2015.

[7] C. Bodenstein, M. Tremer, J. Overhoff, and R. P. Würtz, “A Smartphone-controlled Autonomous Robot,” 2015.

[8] C. Krofitsch, C. Hinger, M. Merdan, and G. Koppensteiner, “Smartphone driven

control of robots for education and research," *Proc. 2013 Int. Conf. Robot. Biomimetics, Intell. Comput. Syst. ROBIONETICS 2013*, no. November, pp. 148–154, 2013.

[9] M. S. Shah and P. B. Borole, "Surveillance and Rescue Robot using Android Smartphone and the Internet," pp. 1526–1530, 2016.

[10] Kwang Yong Lim *et al.*, "A wearable, self-calibrating, wireless sensor network for body motion processing," *2008 IEEE Int. Conf. Robot. Autom.*, pp. 1017–1022, 2008.

[11] M. Bourogaoui, I. Jlassi, S. K. El Khil, and H. Ben Attia Sethom, "An effective encoder fault detection in PMSM drives at different speed ranges," *Proc. - SDEMPED 2015 IEEE 10th Int. Symp. Diagnostics Electr. Mach. Power Electron. Drives*, pp. 90–96, 2015.

[12] D. Kubus, C. Guarino Lo Bianco, and F. M. Wahl, "A sensor fusion approach to improve joint angle and angular rate signals in articulated robots," *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 2736–2741, 2012.

[13] D. Kubus and F. M. Wahl, "A sensor fusion approach to angle and angular rate estimation," *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 2481–2488, 2011.

[14] C. Yang and T. Murakami, "An approach to walking assist control by a multi-legged system in human gait motion," *Proceedings, IECON 2014 - 40th Annu. Conf. IEEE Ind. Electron. Soc.*, pp. 5236–5241, 2014.

[15] Z. Luo, C. K. Lim, I. M. Chen, and S. H. Yeo, "A virtual reality system for arm and hand rehabilitation," *Front. Mech. Eng.*, vol. 6, no. 1, pp. 23–32, 2011.

APPENDIX A

MATLAB Code

```
1     data = 0;
2     bt = Bluetooth('FPGA BLUETOOTH    ',1);     %% Bluetooth Device name
and channel
3     fopen(bt);                                  %% Connect to Bluetooth
device
4     %% Vriable Initialization
5     A = [];
6     T = [];
7     X = 1;
8     fwrite(bt, uint8(240));
9     pause(0.5);
10    fwrite(bt, uint8(240));
11    pause(0.5);
12
13    fwrite(bt, uint8(32));  %% Encoder feedback Controller P gain value
14    fwrite(bt, uint8(11));  %% Encoder feedback Controller P gain set code
15    fwrite(bt, uint8(12));  %% Encoder feedback Controller P gain set code
16
17    fwrite(bt, uint8(4));   %% Encoder feedback Controller i gain value
18    fwrite(bt, uint8(13));  %% Encoder feedback Controller i gain set code
19    fwrite(bt, uint8(14));  %% Encoder feedback Controller i gain set code
20
21    fwrite(bt, uint8(32));  %% P gain Gyroscope feedback controller
22    fwrite(bt, uint8(15));  %% Gyroscope feedback Controller p gain set
code
23    fwrite(bt, uint8(16));  %% Gyroscope feedback Controller p gain set
code
24
25    fwrite(bt, uint8(8));   %% i gain Gyroscope feedback controller
26    fwrite(bt, uint8(17));  %% Gyroscope feedback Controller i gain set
code
27    fwrite(bt, uint8(18));  %% Gyroscope feedback Controller i gain set
code
28
29    fwrite(bt, uint8(110)); %% Motor a rpm speed
30    fwrite(bt, uint8(1));   %% Motor a rpm speed set code
31    fwrite(bt, uint8(2));   %% Motor a rpm speed set code
32
33    fwrite(bt, uint8(110)); %% Motor b rpm speed
34    fwrite(bt, uint8(3));   %% Motor b rpm speed set code
35    fwrite(bt, uint8(4));   %% Motor b rpm speed set code
36
37    fwrite(bt, uint8(0));   %% Encoders set to normal
38    fwrite(bt, uint8(5));   %% Encoders settings function code
39    fwrite(bt, uint8(6));   %% Encoders settings function code
40
41    fwrite(bt, uint8(0));    %% MSB Gyroscope controller angular set point
42    fwrite(bt, uint8(19));   %% Gyroscope controller angular set point
function code
43    fwrite(bt, uint8(20));   %% Gyroscope controller angular set point
function code
44
45    fwrite(bt, uint8(0));    %% MSB Gyroscope controller angular set point
46    fwrite(bt, uint8(21));   %% Gyroscope controller angular set point
function code
47    fwrite(bt, uint8(22));   %% Gyroscope controller angular set point
function code
```

```
48    subLoop = tic;
50    fullLoop = tic;
51
52    for i = 1:1:40              %%Fetch results for 4 seconds
53    fwrite(bt, uint8(240));
54    elTime = toc(subLoop);       %% Timing calculations for improving
system performance
55    if(elTime<.1)
56    pause(0.1-elTime);
57    end
58    subLoop = tic;
59    toc(fullLoop)
60    fullLoop = tic;
61    fwrite(bt, uint8(240));
62    data = fread(bt,16);                %% Read data from Bluetooth Device
63    dataInBinary = (dec2bin(data));
64    A = [A single(data)];
65    T(:,X) = X;
66    X = X + 1;
67    ResultsDLive;                       %% Display updated results
68    fwrite(bt, uint8(0));
69    end
70
71    fwrite(bt, uint8(128)); %% Encoder disconnect setting
72    fwrite(bt, uint8(5));   %% Encoders settings function code function
code
73    fwrite(bt, uint8(6));   %% Encoders settings function code function
code
74
75    for i = 1:1:40            %%Fetch results for 4 seconds
76    %%%%%%%%%%%%%%%
77    end
78
79    fwrite(bt, uint8(4));    %% MSB Gyroscope controller angular set point
80    fwrite(bt, uint8(19));   %% Gyroscope controller angular set point
function code
81    fwrite(bt, uint8(20));   %% Gyroscope controller angular set point
function code
82
83    fwrite(bt, uint8(0));    %% MSB Gyroscope controller angular set point
84    fwrite(bt, uint8(21));   %% Gyroscope controller angular set point
function code
85    fwrite(bt, uint8(22));   %% Gyroscope controller angular set point
function code
86
87    for i = 1:1:70            %Fetch results for 7 seconds
88    %%%%%%%%%%%%%%%
89    end
90
91    fwrite(bt, uint8(26));   %% MSB  of Gyro Scope set point angle
92    fwrite(bt, uint8(19));   %% function code
93    fwrite(bt, uint8(20));   %% function code
94
95    fwrite(bt, uint8(169));  %% LSB gyro set point
96    fwrite(bt, uint8(21));   %% function code
97    fwrite(bt, uint8(22));   %% function code
98
99    for i = 1:1:80
100   %%%%%%%%%%%%%%%
101   end
102
103   pause(0.1);
104   %% set all to default
105   fwrite(bt, uint8(0));   %% Set to 0 (Ecoders to normal)
```

```
106   fwrite(bt, uint8(5));   %% Encoders set code
107   fwrite(bt, uint8(6));   %% Encoders set code
108   fwrite(bt, uint8(0));   %% Motor b rpm speed set to be zero
109   fwrite(bt, uint8(3));   %% Motor b speed set code
110   fwrite(bt, uint8(4));   %% Motor b speed set code
111   fwrite(bt, uint8(0));   %% Motor a speed set to be zero
112   fwrite(bt, uint8(1));   %% Motor a speed set code
113   fwrite(bt, uint8(2));   %% Motor a speed set code
114   pause(0.5);
115   fclose(bt);             %% Disconnect Bluetooth Device
116   clear bt;               %% Clear the bt Variable
117   disp('Bluetooth link released')      %% Display Message that the
Bluetooth Link is released
```

APPENDIX B

Verilog Code Excerpt (Module PI Gyro Feedback controller)

```
delay:
begin
if            (count_reg < 20'b1111_1111_1111_1111_1111)
begin
count_next                        =     count_reg + 1;
state_next                        =     delay;
end
else if (start != 1'b1)
begin
state_next = delay;
error_next                =     0;
propotional_next                  =     0;
integral_next             =     0;
output_value_a_next       =     0;
output_value_b_next       =     0;
end
else
begin
count_next          =     count_reg + 1;
state_next          =     calculate_error;
if(active_motor_hold_reg !=  active_motor_reg)
begin
error_next                =     0;
propotional_next                  =     0;
integral_next             =     0;
output_value_a_next       =     0;
output_value_b_next       =     0;
active_motor_hold_next            =     active_motor_reg;
end
if(set_point > 0)
set_point_reg             =     set_point;
else
set_point_reg = 0;
end
end
calculate_error:
begin
if                  (set_point == process_value)
begin
error_next                                =     0;
propotional_next                  =     0;
integral_next                     =     0;
output_value_a_next       =     0;
output_value_b_next       =     0;
end
else if       (set_point  > process_value)
begin
error_next     = set_point - process_value;
```

```
if(error_reg[12] == 1'b0)
begin
propotional_next              =     error_reg;
integral_next                       =     integral_reg      +
(error_reg[12:5]);
output_value_b_next           =     integral_reg + propotional_reg;
output_value_a_next           =     0;
active_motor_next             =     1'b1;
end
else
begin
process_value_neg             =     13'b1111_1111_1111_1    -
error_reg;
propotional_next              =     process_value_neg;
integral_next                       =     integral_reg      +
(process_value_neg[12:5]);
output_value_b_next           =     0;
output_value_a_next           =     integral_reg + propotional_reg;
active_motor_next             =     1'b0;
end
end
else if (set_point < process_value)
begin
error_next   = process_value - set_point;
if(error_reg[12] == 1'b0)
begin
propotional_next  =     error_reg;
integral_next     =integral_reg     +(error_reg[12:5]);
output_value_a_next           =     integral_reg + propotional_reg;
output_value_b_next           =     0;
active_motor_next             =     1'b0;
end
else
begin
process_value_neg       =     13'b1111_1111_1111_1    -    error_reg;

propotional_next        =     process_value_neg;
integral_next                       =     integral_reg      +
(process_value_neg[12:5]);
output_value_a_next           =     0;
output_value_b_next           =     integral_reg + propotional_reg; -
active_motor_next             =     1'b1;
end
end
else
begin
output_value_a_next           =     output_value_a_reg;
output_value_b_next           =     output_value_b_reg;
end
state_next = output_adjust;
end
output_adjust:
begin
if(count_reg == 3'b000)
begin
if(error_reg > error_hold_reg)
adjust_output_next      =     ~adjust_output_reg;
error_hold_next         =     error_reg;
```

```
count_next                      =     count_reg+1'b1;
end
if(adjust_output_reg == 1'b1)
begin
output_value_a_out_next       =  ~output_value_a_reg; //+ 1'b1;
output_value_b_out_next       =  ~output_value_b_reg;// + 1'b1;
end
else
begin
output_value_a_out_next             =     output_value_a_reg;
output_value_b_out_next             =     output_value_b_reg;
end
count_next = count_reg + 1'b1;
state_next = calculate_output;
end
calculate_output:
                                                  //adjust   output   as
needed  (for  example  if  motor  maximum  power  output  capability  is
reached)
begin
if                (output_value_a_reg[WIDTH-1] == 1'b1)
begin
      output_value_next             = 12'b1111_1111_1111;
      integral_next                 = 12'b1111_1111_1111;
      output_value_reg_out          = 12'b1111_1111_1111;
      overflow_next                 = 1'b1;
end
    count_next                      = 0;
    state_next                      = delay;
end
endcase
```

VITA

Muhammad Ameen Majeed holds a Bachelor of Electronics Engineering, Hamdard University Karachi Pakistan, 2012. He is currently a candidate Master of Science “Power Electronics and Clean Energy Systems” degree under the advisory of Asst. Prof. Dr. Salih Barış Öztürk at Okan University, Istanbul. His areas of interest include Verilog HDL code development, FPGA based digital circuit designs, microcontrollers, clean energy systems and computer vision.

He can be reached in care of:

Asst. Prof. Dr. Salih Barış Öztürk

Department of Electrical and Electronics Engineering

Okan University

Tuzla Campus, Istanbul 0 (216) 677 16 30 #2461