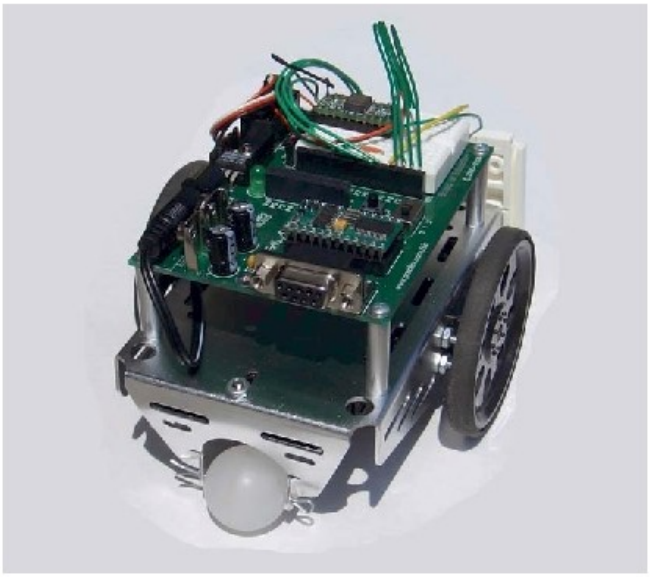
NORTHWESTERN
UNIVERSITY

Dept. of Mechanical Engineering

Robotic Control with Gyroscopic Sensing



ME 224 – Final Project
Prof. Espinosa

Team #1
Stefan Bracher
Jason Lintker
Fabian Wittmer

Table of Contents

Introduction

Each week during this quarter we have conducted labs individually to enhance our skills with LabVIEW and electronic systems. During the final project, we have used the combined knowledge of three team members to create a robot with controlled motion. Our goal was to program a desired route for the robot to follow and use gyroscopic sensing to enable the robot to correct any deviation from that route.

Boe-bot Assembly

Basic Components

The original Boe-bot kit consists of two servo motors and a printed circuit board attached to an aluminum chassis. A breadboard provides a convenient interface for additional electronic devices. This very simple structure allows users to customize the robot design with a variety of features and components.

BASIC Stamp

The BASIC Stamp 2 module contains its own processor, memory, clock, and an interface with 16 I/O pins. The BASIC Stamp program we created (see A.2) and wrote onto the module allowed us to define a specific route for the robot to follow. Additional features of the program enable the robot to sense any deviation from the desired direction. The robot then responds with a small rotation to correct the error.

Servos

Two small servo motors are attached to the wheels to drive the robot with a high degree of accuracy. Servos use built in circuitry to monitor the signal from a potentiometer which is connected to the output shaft. In this way, the servo can control the angle of the output shaft at all times. This angle is controlled with coded signals sent from the circuitry of the robot. Changes in the signal cause changes in the angular position of the shaft, creating motion of the robot. This type of servo control is referred to as Pulse Coded Modulation. In basic terms, the length of the pulse determines the direction and speed of rotation.

Gyroscope

Overview

A classical gyroscope is a device consisting of a spinning mass, typically a disk or wheel, mounted on a base so that its axis can turn freely in one or more directions and thereby maintain its orientation regardless of any movement of the

base. When spinning, the gyroscope has special properties. Many spinning objects exhibit some of these properties; the rotation of the earth about its axis gives it the properties of a huge gyroscope. Once a gyroscope starts to spin, it will resist changes in the orientation of its spin axis. For example, a spinning top resists toppling over, thus keeping its spin axis vertical. If a torque, or twisting force, is applied to the spin axis, the axis will not turn in the direction of the torque, but will instead move in a direction perpendicular to it. This motion is called precession.
The upcoming technology, however, is part of the Micro-Electro-Mechanical Systems family (MEMS). This implementation does consist of a vibrating mass rather than a spinning one. The principle used to measure the angular acceleration is the Coriolis effect. Whenever a mass is moved in a rotating system a force act on the mass. This force is called Coriolis force and is proportional to the angular velocity of the system and the velocity of the mass.
MEMS gyroscopes are already used in the car industry for navigation systems.

Implementation

The gyroscope used in this project is the ADXRS150ABG, built by Analog Devices. It induces a voltage signal proportional to the angular rate of change. Up to 150 degrees per second are detectable. The power supply is a 5 Volt DC signal. The output ranges from 0.25V to 4.75V. No motion equals 2.5V
In order to check the gyroscope is working, we hooked it up to an oscilloscope. In the resting state we got a signal of 2.48V. When turning the gyroscope, the signal in- or decreased depending on the direction.
Usually, a calibration is needed for further operations. This means the gyroscope is spun at a different velocities, simultaneously the voltage output is acquired. With a curve fitting tool one gets the voltage in dependency of the angular rate of change.
However, for several reasons we decided on not to do a classical calibration. First of all, neither the absolute voltage value nor the angular rate of change is of interest in

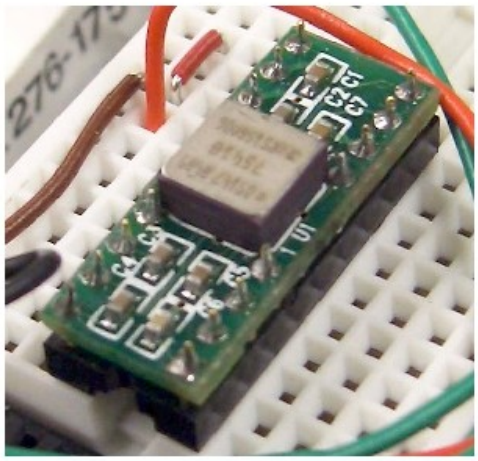


Fig. 2 Gyroscope

this project. We are only interested in the angle. So, the angular rate of change needs to be integrated once. By calibrating before the integration errors would be summed up and propagate. Therefore a calibration with respect to the angle is reasonable.
In the numerical integration approach we use, a constant time increment of one unit is assumed. Therefore, the integrated value depends on the frequency of data acquisition. The frequency in turn depends on the amount of code in between each cycle. This means that a calibration of the gyroscope is not reasonable until the final version is ready. Only at this point the parameters can be adjusted.

A/D Converter

Overview

An A/D Converter converts a voltage in a digital number. We used an ADC0804LCN, as our basic stamp had no A/D converter included (just digital inputs).

Specifications of the ADC0803LCN:

- 8bit converter
- Can be used with internal or external clock
- Analog input range 0 V to VCC
- Single 5 V supply "
- Guaranteed specification with 1 MHz clock

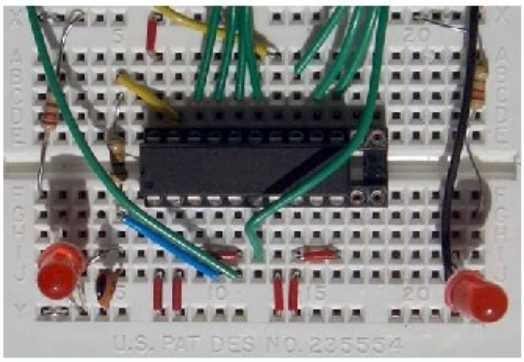


Fig. 3 A/D converter

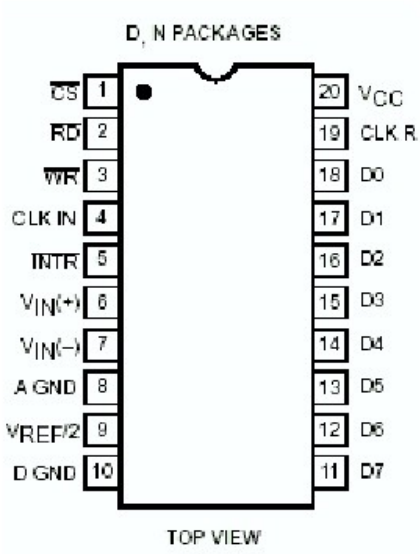


Fig. 4 Pins of the ADC0804LCN

Function

The conversion is started with a pulse at the WR pin, and the CS pin on low. As it is an 8bit converter, VCC will be represented by the number 256 and 0V with 0. When the conversion is complete, the INTR pin will make a high-low conversion, what could be used as an interrupt for a processor. Setting the RD and CS pin to low will clear the interrupt and enable the output pins D0-D7 for reading.

Implementation

The ADC0804LCN can be used in several different modes. However we decided that continuous conversion suits our purposes the best.
The following pin connection was supposed by the spec sheet:

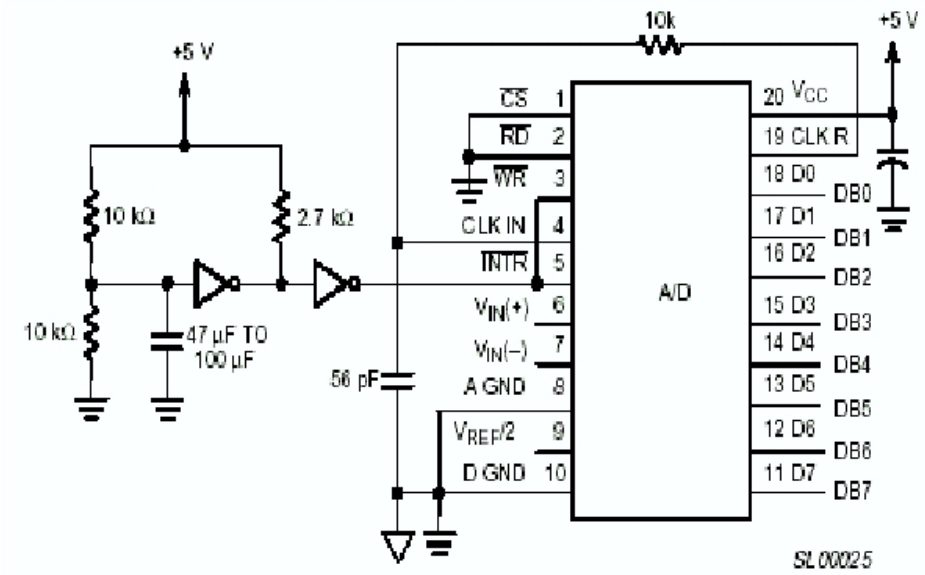


Fig. 5 Continuous mode connection

The output pins D0-D7 with 0V as logical low and 5V as logical high (what corresponds to the specifications of the basic stamp) are connected to the basic stamp.
Setting the CS and RD pins on permanent low will always clear the interrupt and enable the output pins for reading. To be able to use the internal clock, CLK R

has to be connected with a 10k resistor to CLK IN and CLK IN with a 56pF compensator to ground.
To start a conversion, there has to be a pulse on the WR pin. Instead of the logic on the left side of the spec sheet, we use the basic stamp to send this pulse.

Computer Programming

LabVIEW – A/D Conversion

LabVIEW – A/D Conversion

For there were some difficulties with the acquisition of the 8bit A/D converter, a temporary solution was needed urgently. We opted for a software based LabVIEW solution. The gyroscope yields an analog signal ranging from 0.25 to 4.75 Volts. The DAQ board has a higher resolution than 8bit and represents 5V with 2048. Therefore the read value has to be divided by 8 to convert it to 8bit.

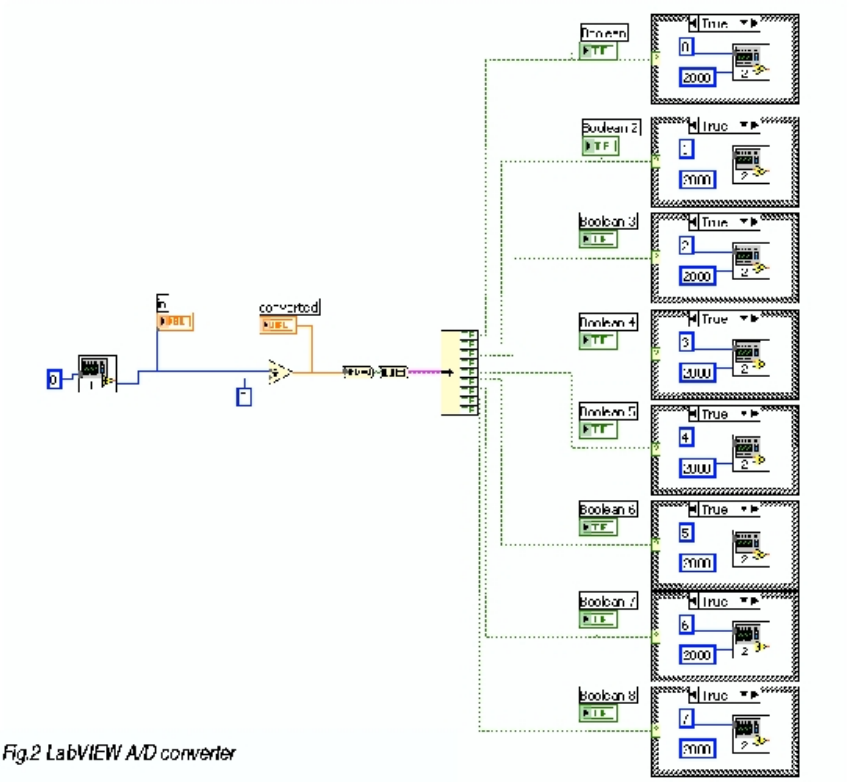
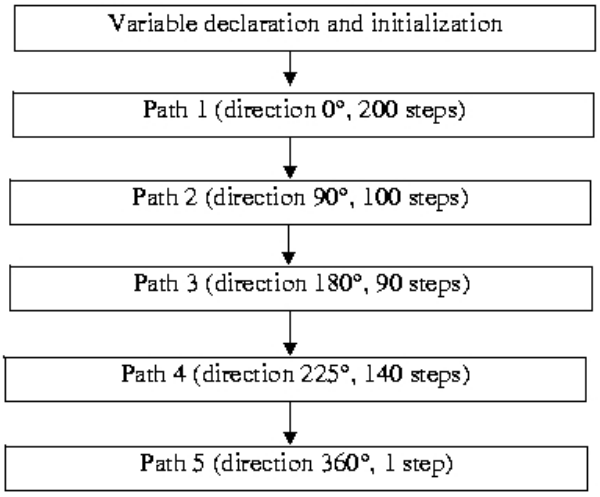


Fig.2 LabVIEW A/D converter

We then used the LabVIEW function "number to Boolean array" which yields eight true false statements. Each statement is the input of a true/false case structure which sets the respective port on the DAQ board to low or high. Those eight ports equal the outputs of a hardware based A/D converter.

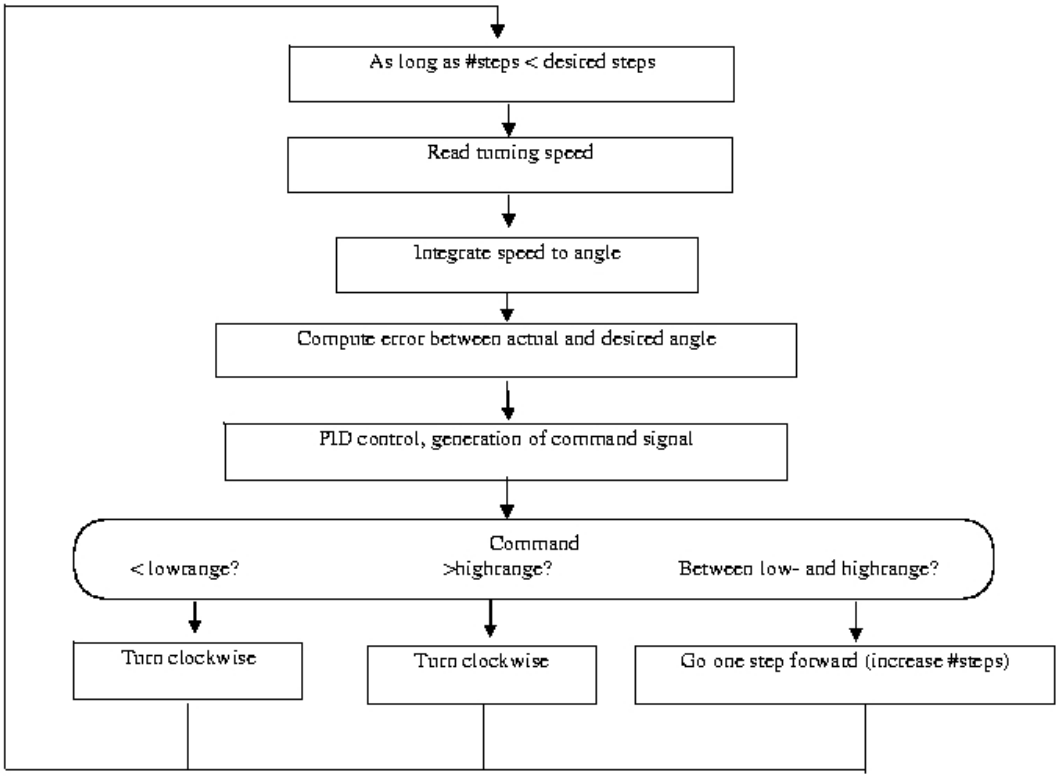
BasicStamp Programming

Structure of the program



The program begins with the Variable declaration and initialization followed by sequences for the different paths to take. Each path sequence specifies the (absolute) direction and the number of steps to go and calls the subroutine "coordinates" to execute these paths.

The feedback control of the gyro is implemented in the coordinate subroutine, the robot can only make a step forward, if the error between actual and desired angle is very small. As soon as the error exceeds a certain amount, the robot has to correct its path by turning clock, or counterclockwise.



Program code

Compiler information:

'{$STAMP BS2}	*Tells the compiler that we use the Basic Stamp2*
'{$PBASIC 2.5}	Specifies the program language used: PBASIC 2.5

Variable declaration:

speed VAR Word	The turning speed
angle VAR Word	The absolute angle
steps VAR Word	Number of steps to take
direction VAR Word	Direction (absolute angle) to take
error VAR Word	The error between desired and actual direction
errorsum VAR Word	the sum of the error
errordiv VAR Word	the change of the error
lasterror VAR Word	the last error before the current one
i VAR Word	a counting variable for steps
command VAR Word	the command produced by the PID controller

```
lowrange CON 32000          A variable used for the control (see later)
highrange CON 32200         An other variable used for the control
offset VAR Word             The offset of the gyroscope at zero speed
```

Initialization of the variables:

```
angle=0
lasterror=0
error=0
errorsum=0
i=0
LOW 11                      A pulse is sent to the A/D converter to start the
conversion
PAUSE 200
HIGH 11
PAUSE 200
offset=IN0+(IN1*2)+(IN2*4)+(IN3*8)+(IN4*16)+(IN5*32)+(IN6*64)+(IN7*128)
                            The zero speed offset of the gyroscope is read
(At zero speed the gyro produces around 2.5V)
LOW 11

PAUSE 1000
```

The main program

```
'path 1
steps=200                   Go 200 steps
direction=0                 go in direction 0 (=0°)
GOSUB coordinates           call of the subroutine "coordinates" to do the path

'path 2
i=0                         reset number of steps
steps=100                   go 100 steps
direction=4875              go in direction 4857 (Experimentally determined to be 90°)
GOSUB coordinates           call of the subroutine "coordinates" to do the path

'path 3
i=0                         reset number of steps
steps=90                    go 90 steps
direction=9750              go in direction 9750 (Experimentally determined to be 180°)
GOSUB coordinates           call of the subroutine "coordinates" to do the path

'path 4
i=0                         reset number of steps
steps=140                   go 140 steps
direction=12188             go in direction 12188 (Experimentally determined to be 225°)
GOSUB coordinates           call of the subroutine "coordinates" to do the path

'path 5
i=0                         reset number of steps
steps=1                     go 1 step (just turn)
direction=19000             go in direction 19000 (Experimentally determined to be 360°)
GOSUB coordinates           call of the subroutine "coordinates" to do the path

END
```

coordinates subroutine

```
coordinates:
resume:                          jump marker
DO WHILE (i<=steps)              as long as steps<i

GOSUB ReadValue                  read the speed
error=direction-angle            compute the error
'errorsum=errorsum+error         compute the errorsum (fot the I part of PID, not used)
'errordiv=lasterror-error        compute the errordifference (for the D part of PID, nor
used)'lasterror=error

'PID control
command=(2*error)'-(2*errordiv)'+(1*errorsum)   PID control computates command signal
(Originally we wanted to implement full PID, but P turned out
        to be good enough)

IF ABS(command)<=100 THEN gostraight      if command small call subroutine go straight
IF (command+32100)<lowrange THEN turnccw  command < lowrange call turnccw*
IF (command+32100)>highrange THEN turncw  command >highrange call turncw*

*The addition of 32100 is used to avoid problem with negative numbers

LOOP
RETURN
```

gostraight subroutine

```
PULSOUT 13, 791                  send forward pulse to left wheel
LOW 13
PULSOUT 12, 737                  send forward pulse to right wheel
LOW 12
PAUSE 20
i=i+1                            increment the number of steps taken
GOTO resume                      jump to resume (in coordinates subroutine)
```

turnccw subroutine:

```
PULSOUT 12, 745                  send forward pulse to right wheel
LOW 12
PULSOUT 13, 748                  send backward pulse to left wheel
LOW 13
GOTO resume                      jump to resume (in coordinates subroutine)
```

turncw subroutine:

```
PULSOUT 12, 775                  send backward pulse to right wheel
LOW 12
PULSOUT 13, 785                  send backward pulse to left wheel
LOW 13
GOTO resume                      jump to resume (in coordinates subroutine)
```

ReadValue subroutine:

```
HIGH 11                                  Start AD conversion
speed=IN0+(IN1*2)+(IN2*4)+(IN3*8)+(IN4*16)+(IN5*32)+(IN6*64)+(IN7*128)-offset

calculate the speed

IF NOT ABS(speed)<2 THEN                 suppresses small (error) signals
angle=angle+speed                        integrate the angle
ENDIF

LOW 11
```

Testing

A great deal of testing was necessary to perfect the parameters of our BASIC Stamp programming. Modifications were made to the pulse trains to ensure that the servos could create both straight-line motion and smooth turning motion in both directions.

Testing was also vital to the creation of a reliable control loop. We discovered that although a control algorithm may work well in theory, it may not perform well in actual tests. By testing a variety of different algorithms and adjusting the error sensitivity, we were able to create a very stable control system for our robot.

Challenges

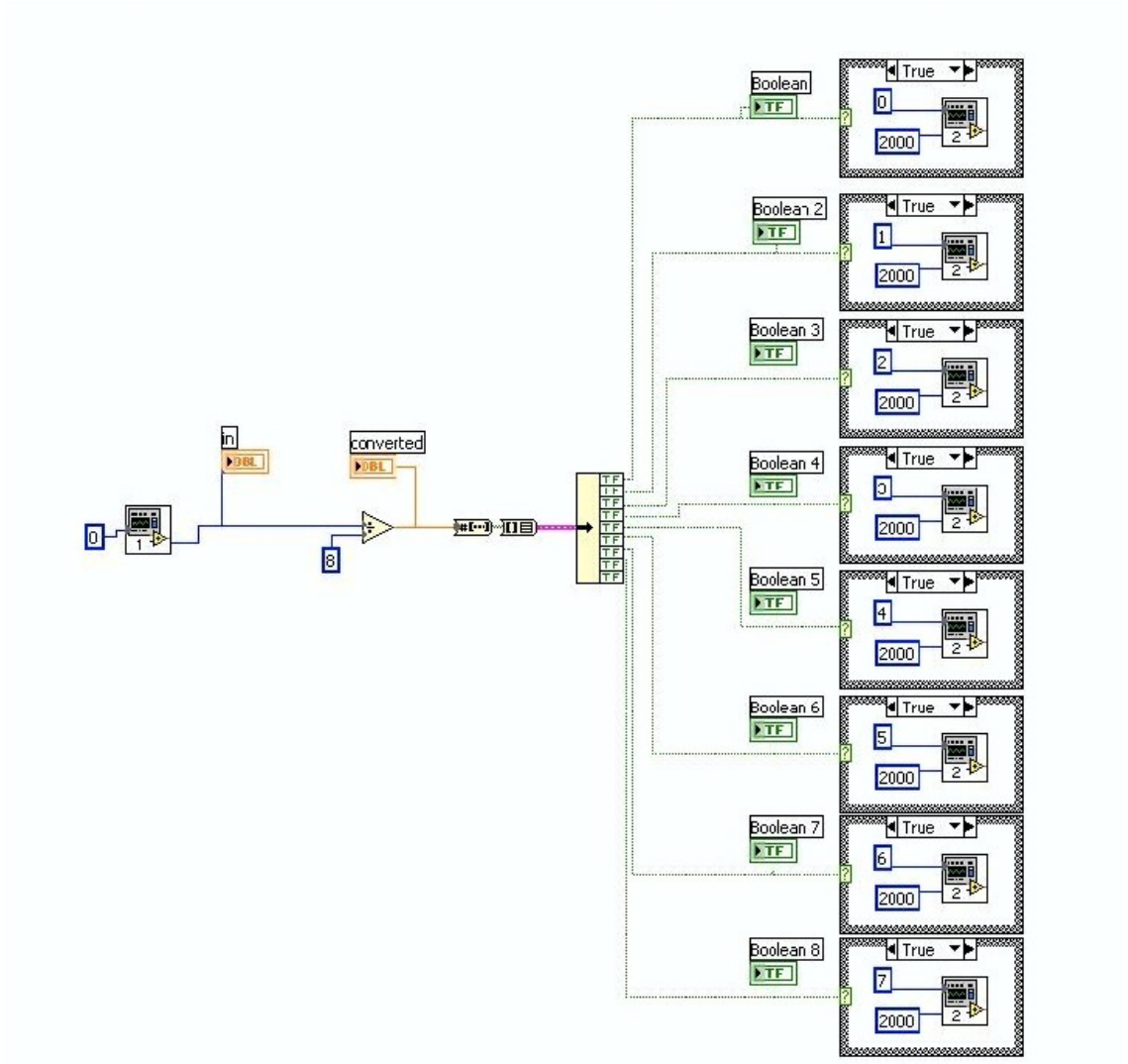
The primary challenge faced by our team was the implementation of gyroscopic control. In the early stages of the project, we attempted to use several different control algorithms in our BASIC Stamp programming. The first version of our control loop was a simple proportional control (error = setpoint – measurement). While it was obvious that the logic behind this control was sound, the application to our robot created some problems. When attempting to correct for error, the robot would often overshoot the target direction. The addition of a derivative control, and eventually a complete PID control, didn't offer much improvement. The problem was finally eliminated after fine-tuning the turning speed and error range.

An additional challenge faced by our team was the addition of the A/D converter to the robot circuitry. While the addition of the converter allowed our robot to operate independently from LabVIEW and the associated cables, the installation was somewhat complicated. An additional breadboard was attached to the chassis to create adequate room for the chip and the additional wiring. Since the specifications for the chip did not provide a clear wiring guide, several wiring variations were constructed before the chip could properly convert the signal.

Conclusions

In conclusion, our team has clearly demonstrated the ability to create a functional system for gyroscopic control on a robot. Through the integration of a MEMS gyroscope and BASIC Stamp programming, our robot was able to successfully follow a specified route. Additionally, the robot was able to constantly monitor feedback from the gyroscope to correct for any deviation from the desired path. Overall, our team is very satisfied with the accuracy of our control system and the performance of our robot.

Appendix A – Labview



Appendix B – Basic Stamp

```
'{$STAMP BS2}
'{$PBASIC 2.5}


'Variables
speed VAR Word
angle VAR Word
steps VAR Word
direction VAR Word
error VAR Word
errorsum VAR Word
errordiv VAR Word
lasterror VAR Word
i VAR Word
command VAR Word
lowrange CON 32000
highrange CON 32200
offset VAR Word


'DEBUG "start"

'Initialisation
angle=0
lasterror=0
error=0
errorsum=0
i=0
LOW 11
PAUSE 200
HIGH 11
PAUSE 200
offset=IN0+(IN1*2)+(IN2*4)+(IN3*8)+(IN4*16)+(IN5*32)+(IN6*64)+(IN7*128)
LOW 11

PAUSE 1000

'main programm


'path 1
steps=200
```

```
direction=0
GOSUB coordinates

'path 2
i=0
steps=100
direction=4875
GOSUB coordinates


'path 3
i=0
steps=90
direction=9750
GOSUB coordinates

'path 4
i=0
steps=140
direction=12188
GOSUB coordinates

'path 5
i=0
steps=1
direction=19000
GOSUB coordinates

END

'subroutine
coordinates:
resume:
DO WHILE (i<=steps)
'DEBUG SDEC ? i
'DEBUG SDEC ? angle
'DEBUG SDEC ? error
'DEBUG SDEC ? offset
'DEBUG SDEC ? speed

'read angle
GOSUB ReadValue
'compute the error
error=direction-angle
'errorsum=errorsum+error
errordiv=lasterror-error
```

```
lasterror=error

'P control
command=(2*error)'-(2*errordiv)'+(1*errorsum)



'cases
IF ABS(command)<=100 THEN gostraight    'go straight
IF (command+32100)<lowrange THEN turnccw 'turn left
IF (command+32100)>highrange THEN turncw 'turn right



LOOP

RETURN


gostraight:
PULSOUT 13, 791 'left weel
LOW 13
PULSOUT 12, 737 'right weel
LOW 12
PAUSE 20
i=i+1
GOTO resume



turnccw:
PULSOUT 12, 745
LOW 12
PULSOUT 13, 748
LOW 13
'DEBUG "counterclockwise"
GOTO resume

turncw:
PULSOUT 12, 775
LOW 12
PULSOUT 13, 785
LOW 13
'DEBUG "clockwise"
GOTO resume
```

```
ReadValue:
HIGH 11
speed=IN0+(IN1*2)+(IN2*4)+(IN3*8)+(IN4*16)+(IN5*32)+(IN6*64)+(IN7*128)-
offset

IF ABS(speed)<2 THEN
'offset=offset+speed
ELSE
angle=angle+speed
ENDIF

LOW 11
RETURN
```