Universität Bremen

Mikrosensoren, -aktoren und -systeme (IMSAS)

Prof. Dr.-Ing. Walter Lang

Thesis

Automated Calibration for Wireless Flow Sensors

of

Chanaka Lloyd

30th August 2010

Supervised by

Prof. Dr.-Ing. Walter Lang

Dr.-Ing. Reiner Jedermann

DECLARATION

Ich versichere, daß die vorliegende Arbeit — bis auf die offizielle Betreuung durch den Lehrstuhl — ohne Fremde Hilfe von mir durchgeführt wurde. Die verwendete Literatur ist im Literaturverzeichnis vollständig angegeben.

I certify that I have conducted this work on my own and no other supporting material has been used other than those which are listed as references.

Bremen, 30th August 2010

Chanaka Prasaraka Lloyd

ACKNOWLEDGMENT

First and foremost, I am most grateful for my supervisors: Prof. Walter Lang, Director, IMSAS, University of Bremen for his patronage; and Reiner Jedermann, my second supervisor, for assisting me throughout with invaluable engineering advice and having accompanied me with practical implementations of the project.

In addition, I wish to pay my gratitude to several staff members of IMSAS for their assistance in various affairs: Christoph Sosna and Miron Kropp in thermal sensors; Adam Sklorz in LabVIEW® and UltiBoard®; Dieter Gauck and Darren Gauld in software and operating system installations; Cord Winkelmann in SolidEdge®; Gabriel Büschel in admistrative help; and, Alexander Wessels. Additionally, I am thankful for the coding advice from Alfonso Cardell Bilbao of SensiAll, Canary Islands, Spain.

My special thanks goes to Sandeep Maane, Pawel Fulmyk, Javier Palafox, and Zuolin Xu for their invaluable help and tolerance in assisting me in testing the sensor calibration during night and day.

Also, I wish to thank the German Research Foundation (DFG) for having funded this research project as part of the Collaborative Research Centre 637. LaTex, LyX, and Ubuntu were used for this thesis and am delighted to thank those organizations and contributors.

Finally, I am warmed by the support given by my family and my girl friend, Zuolin.

It would not have been possible to have the project completed successfully without the assistance of the aforesaid personnel.

ABSTRACT

This thesis presents the work conducted as an advanced extension to the project work *Integrating Thermal Flow Sensor to TelosB Module*, presented on 6th May 2009 by the author of this thesis report. The added features are autonomous calibration and temperature compensation for thermal flow sensor, a new air channel design, updated electronics for the amplifier circuit, calibration method and results, and a proposal for a sophisticated test bed for future calibration.

The report presents a methodology for autonomous calibration and compensation for the TS10_1 type thermal flow sensor to measure the air flow speed. The design of software and hardware incorporates provision to integrate the work to the *Intelligent Container* project. Automated calibration, dependent on varying ambient temperature, is carried out with a digital potentiometer. Manual Switching technique used in the previous work is also digitized. Therefore, the complete system is fully automated to measure the air flow speeds and radio back to a LabVIEW program in the host computer. Calibration of the thermal sensor is achieved by simulating air flow of different speeds in the range of 1 to 5 m/s using a remote controlled car, on which the sensor is mounted.

This embedded system is based on TinyOS on TelosB wireless sensor nodes, and the programming is done with nesC language. WSN node programming and the circuit design is optimized to consume as less energy as possible in accordance with the stringent requirements of the Intelligent Container project.

Contents

List of Figures

List of Tables

1 INTRODUCTION

The embedded systems have become a day-to-day term among the engineering and scientific communities. It is a discipline that has seen heavy investment and research go in to it during the last three decades. As a result, embedded systems are playing a major role in consumer electronics, bringing quality of life to masses.

Wireless Sensor Networks (WSN) is as fast-developed discipline as embedded systems. WSNs are a necessity than a luxury in all communities around the world, and they are heavily used in all aspects of human life. Radio communication enables the sensor output to be monitored in real-time—or be stored for later use—which makes the whole idea of *sensing* a novelty. Such a network of communication among active nodes gives way to providing intelligent solutions for myriad of problems, ranging from domestic to commerce.

An integrated platform of these two pervasive technologies in tandem makes even a stronger platform on which many solutions can be developed. One such platforms is *TelosB* from Crossbow Technology Inc^{TM}. There are other such platforms such as Meshbean, iMote, TinyNode, Micaz, Iris, Eyes, etc. The work in this thesis uses TelosB as the main platform for its versatility and good overall performance; however, Meshbean platform is proposed for future deployment for its superior radio communication strength.

The Intelligent Container Project is a research attempt at bettering the ambient conditions during transport to improve the quality of perishable foods and extending shelf life times. IMSAS plays a pioneering role in the aforesaid project among other partners of Microsystems Center Bremen (MCB). The Intelligent Container project requires the assistance of several sensors, such as temperature, humidity, flow, and ethylene, to monitor the ambient conditions. Thermal flow sensor (1.2) is the subject of this thesis; electronics are developed to amplify the flow sensor signal and calibrate it autonomously. This work carries far advanced software algorithms and digitized electronics—plus extensive testing for the calibration process—over the mini project *Integrating Thermal Flow Sensor to TelosB Module*[1].

1.1 Intelligent Container Project

The following is an excerpt from the Intelligent Container project website.

"*The Microsystems Center Bremen developed the 'Intelligent Container' as an autonomous transport monitoring system for perishable and sensitive goods. The systems links technologies from the fields of RFID, sensor networks, and software agents to provide a permanent and freight-specific supervision of each transport package along the supply chain. Local pre-processing of sensor information reduces costs for external mobile communication. A quality prediction model runs on an embedded processor platform that is integrated into the container, truck or semi-trailer. If a risk for the quality of loaded freight items is detected, the autonomous supervision system sends a warning message to the transport operator*[2]."

One future goal of the project is air flow monitoring within the sea freights. The importance of air flow monitoring stems from the stagnating air pockets within a freight that gives way to temperature rise, hence reducing the quality and the shelf life of perishable foods under transportation. Such a loss amounts to millions of dollars annually around the world. Indeed the Intelligent Container project is already capable of identifying such abnormal temperatures within a container; however, it lacks the assistance of air flow sensors to determine if such a rise in temperature is due to elevated enzyme activity within foods or cold air not reaching certain areas of the container.

The air flow sensors alleviate this problem altogether by monitoring different sections of the container periodically by measuring the air flow speeds. If the container shows such stagnant air pockets, a warning message can be issued to warn the officers on deck and to the shippers of the goods. Either the arrangement of food pallets within the container can be rearranged (where possible) or by learning from that particular incident, future containers can be packed allowing enough space for the cold air from the *reefer unit*[1] to reach warmer areas.

The following figure elaborates the above point. The figure below shows surface temperature variation on a truck container, partially loaded with frozen foods. The refrigeration unit is attached just below the ceiling, and the fan direction is marked on the top surface of the figure.

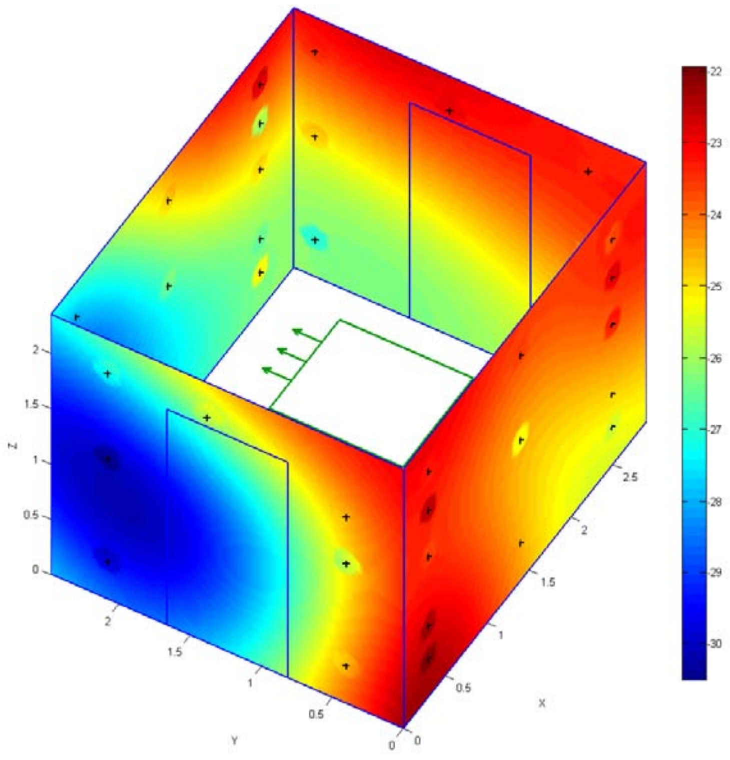


Figure 1: Spatial temperature variation inside a container

Incorrect arrangement of pallets could produce a temperature variation profile as above. A separate study conducted in IMSAS allows the shippers of goods to visualize how the pallets are packed inside a container [3]. Therefore, the temperature variation profile, visual of the pallet arrangement, and flow sensor data give better understanding of the prevalent situation. That enables the crew to take appropriate action to prevent temperature anomalies and save the foods from perishing prematurely.

1.2 Flow Sensor

The thermal flow sensor is a product of IMSAS. It is manufactured by a high temperature fabrication process, and its measurement methodology is based on a combination of two thermopiles (made of p-doped polysilicon and WTi) and a heater (WTi). The LPCVD (Low Pressure Chemical Vapor Deposition) passivation layer of these sensors makes it more robust towards liquid and gaseous applications. There are 15 thermocouples on each thermopile. Each thermocouple is positioned at close range to the heater. The heater is powered up and it emits heat to the measured medium, be it gas or liquid. When the medium is allowed to pass over the sensor, the thermopiles measure the temperature profile (i. accumulated voltages of all thermocouples) of the medium with the use of its thermocouples. The measured temperature profile is dependent on the type of medium and the rate of flow. In effect, the measurement of temperature profile means the measurement of the end-to-end, accumulated voltage difference of each thermopile. The

[1]Reefer unit is the refrigeration unit attached to each container during transportation to maintain a desired temperature.

voltage difference of the two thermopiles vs. flow rate characteristics enable measurement of flow rate [1, 4, 5, 6].
The following is a photographed thermal flow sensor.

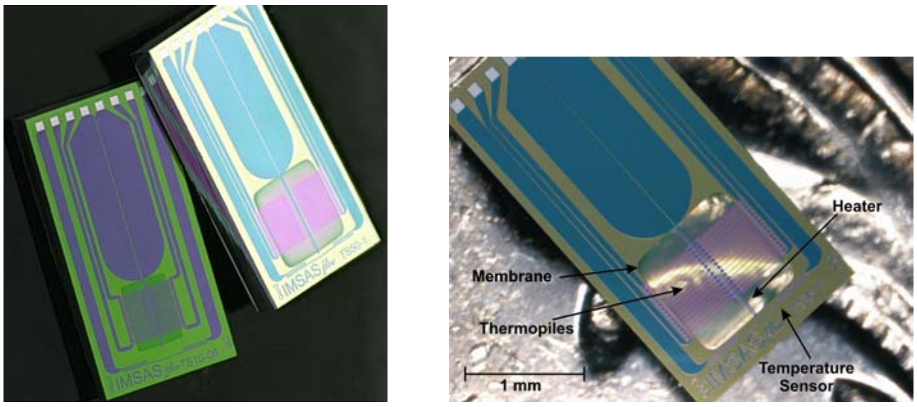


Figure 2: Flow sensor (*Source: IMSAS/MCB*)

When air passes over the flow sensor, it produces a maximum voltage difference signal of +/- 45 mV[2]. The measured laboratory values for the voltage output of each thermopile—with air as the medium and at 50^{o}C—was approx. 82 mV. The following figure on the left (at zero flow rate, but heater powered up) illustrates the temperature profile concept described above; the flow sensor of type *TS10_1* used in this study with its air channel is shown on the right.

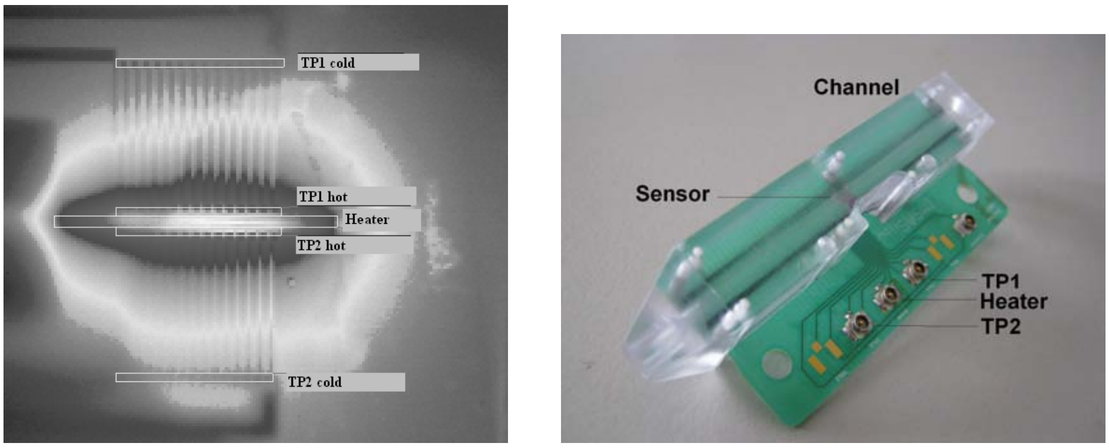


Figure 3: Left: Temperature profile of the flow sensor; Right: The flow sensor with its channel

The air channel design shown above is a new design and is covered in Chapter 3.2.1.

1.3 Previous Work

The mini-project work, stated earlier, by the author involved basic integration of the thermal flow sensor to the TelosB node. Integration was done with a feed-forward differential amplifier circuit; additionally, the circuitry involved methods for manual bridge[3] balancing and temperature compensation (Appendix I, Figure 6). The programmed node reads the amplifier signal and transmits it to a base-station. However, two main factors prevents it from deployment as a prototype for flow measurement in containers.

[2]Voltage difference depends on many factors: temperature, medium, distance to the heater, etc. However, this laboratory value is for air as medium, as measured previously by Christoph Sosna, IMSAS.
[3]Here, bridge refers to a wheatstone bridge.

1. Uncalibrated flow sensor.
2. Manual bridge balancing and compensation.

For successful deployment of the flow sensor in the Intelligent Container project requires much more development than above. However, overcoming the above can be classified as the ***minimum criteria*** for prototype deployment.

1.4 Requirement and Task

The principle requirement of the project is to satisfy the minimum criteria stated in previous section. In addition, with the accumulated experience during the test phase of the project, an advanced testbed for future calibration and an algorithm for the deployment of a *flow prototype* in the Intelligent Container project is proposed. The tasks in detail are as follows.

1. Autonomous balancing of the wheatstone bridge.
2. Autonomous temperature compensation of the bridge for 25K above the ambient temperature (50^oC).
3. New air channel design.
4. Prototype PCB board for the electronics with provision for additional component integration in future.
5. Formulation of a testbed for calibration of the thermal flow sensor.
6. LabVIEW application to display the flow speed (General Application) and to record data in calibration tests (Test Application).
7. Matlab application for calibration test data extraction, manipulation, and analysis.
8. Programming of TelosB motes to sample flow measurement and to facilitate (1) and (2) above.

1.5 Chapter Breakdown and Reference Guide

The following figure of nested rectangles represents the entire structure of the thesis. Each rectangle is representative of individual chapters. The chapter titles are placed just inside the top margin of the rectangles, whereas the short phrases seen inside the left margin of each rectangle means to provide a very concise description of each chapter.

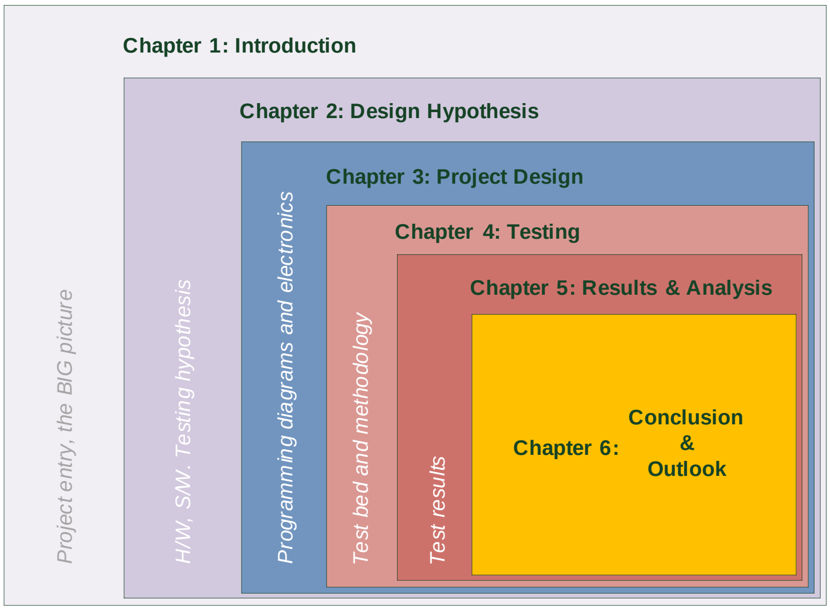

Chapter 1: Introduction
Chapter 2: Design Hypothesis
Chapter 3: Project Design
Chapter 4: Testing
Chapter 5: Results & Analysis
Chapter 6:
Conclusion & Outlook
Project entry, the BIG picture
H/W, S/W. Testing hypothesis
Programming diagrams and electronics
Test bed and methodology
Test results

Figure 4: Chapter breakdown

Chapter 1 introduces the backdrop of the previous work leading up to this work, the Intelligent Container project, the flow sensor, and the tasks fulfilled. Chapter 2 then presents the hardware, software, and testing tasks in conjecture form. This hypothetical description of the entire project design—with many pictorial depictions—is meant to familiarize the reader with the *BIG picture*. Then comes the detailed descriptions of the previously described hypothetical models. Chapter 3 also illustrates the bridge balancing technique and the working principle of the embedded system designed. Chapter 4 deals with all aspects related to the tests conducted in order to calibrate the flow sensor. It also carries a proposal for an advanced testbed model for more precise calibration. Chapter 5 illustrates all test result data, energy consumption aspects, and some analysis of the results. Conclusion and the outlook of the work is discussed in Chapter 6. Then follows the Bibliography and the Appendix. The appendices carry the bulk of the programming code among other supplementary data of relevance to the project.

A data Compact Disk (CD) is also provided with this thesis with all data files, figures, and code.

2 DESIGN HYPOTHESIS

This chapter compounds all software, hardware and testing used in the actual project in to a complete hypothetical model. Therefore, the different units of the sections to follow is discussed in abstraction, leaving out the seemingly complicated details. However, all details are discussed in depth in Chapter 3.

2.1 The Main Target

Among all the tasks listed in the introduction section, one stands out as the main target of this master thesis. It is described below.

The differential voltage of the flow sensor varies with different air flow speeds. Therefore, the main target is to calibrate the sensor under different ambient temperatures to find a relation between the amplified voltage difference and the air flow speed. It is visualized in figure below. Here, the differential voltage is the difference of voltage between the two thermopiles of the flow sensor.

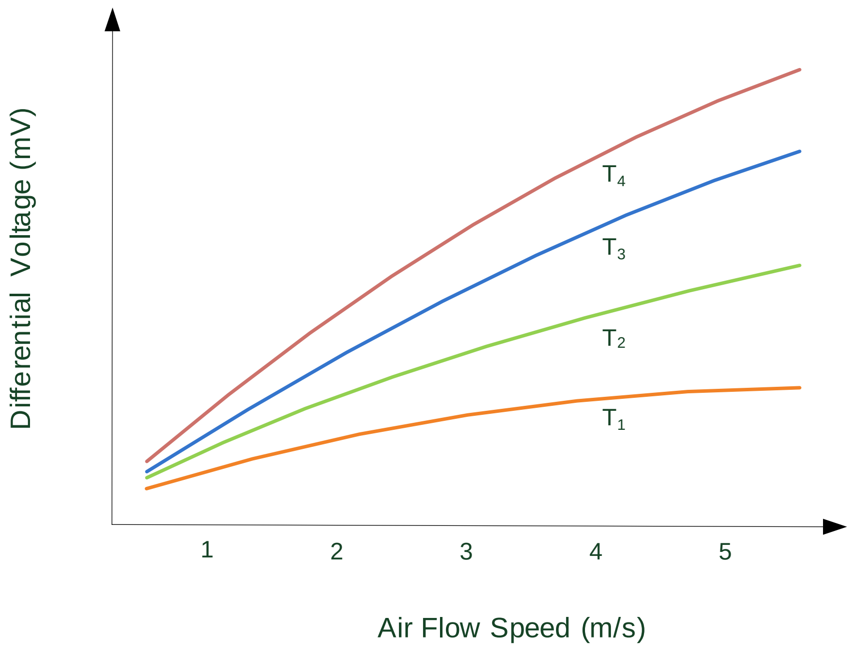


Figure 5: Hypothetical Differential Voltage vs. Average Air Flow Speed

Mathematically, the relation for a certain temperature T_i can be depicted as follows:

$$\triangle V = f^i\left(\bar{\nu}\right) \tag{1}$$

$\triangle V$ is the thermopile voltage difference, and $\bar{\nu}$ is the air flow speed. Determining this function f for T_1, T_2, T_3... for the given flow sensor is the main target.

2.2 Overall Design Hypothesis

The following diagram illustrates all individual components and entities of the involved electronics. It also portrays the software applications which read and process the signals from circuitry.

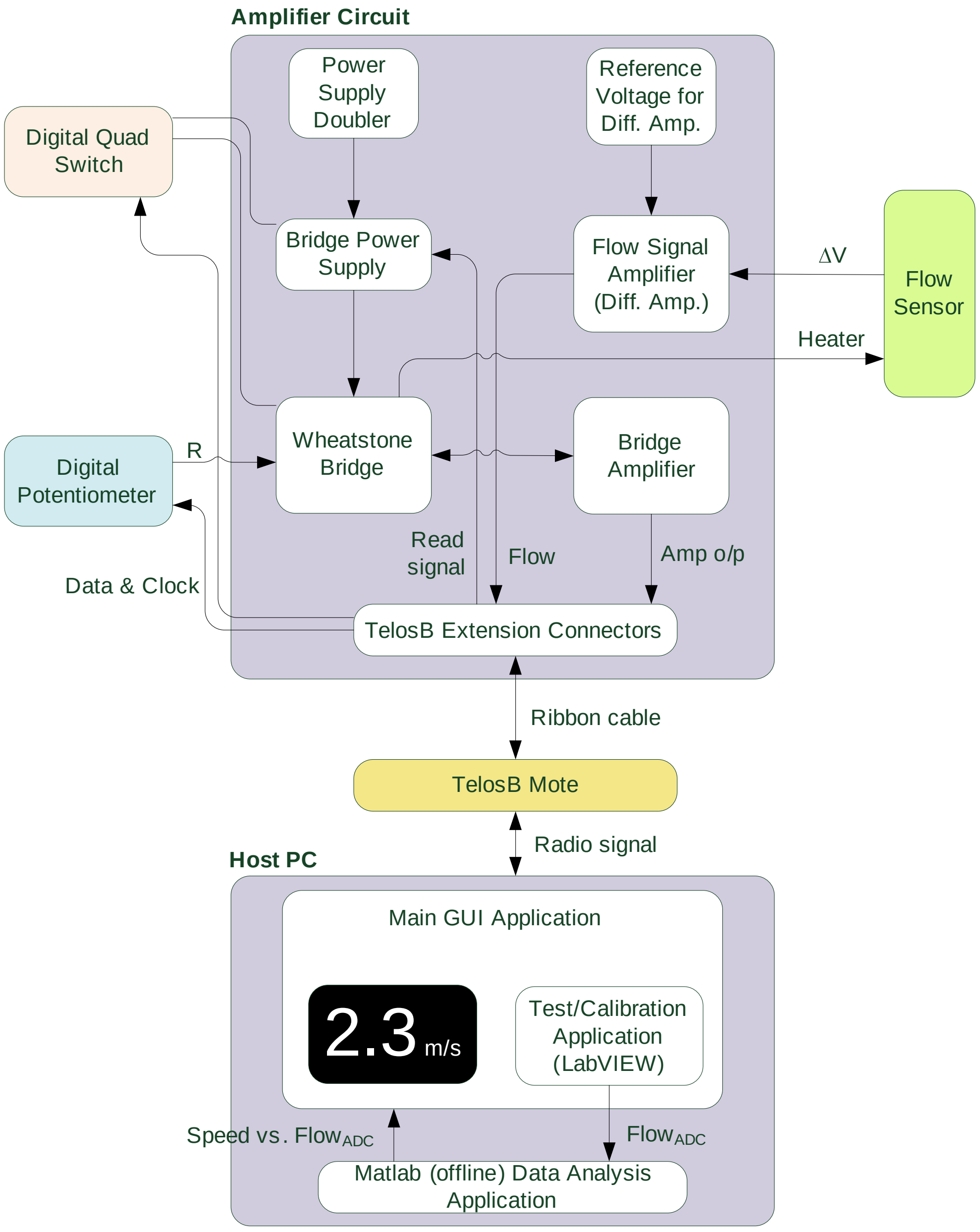


Figure 6: Overall Design

The arrows lines in the above diagram are indicative of data directions, power connections, and communication signals. The rectangle boxes nested within the main components carry descriptions, which are self explanatory. The complete design consists mainly of 6 components:

1. Amplifier Circuit
   This is largely the circuit design carried forward from the mini-project (Chapter 1.3), except the manual potentiometer and jumper settings. It is important to note that one of Digital Potentiometer's lower branches is the heater of the Flow Sensor.

2. Digital Quad Switch
   There are 4 2-way switches in this component. The circuit design uses 3 of them: one each for Bridge Power Supply, Wheatstone Bridge, and Bridge Amplifier (see Figure above). This replaces the manual switching performed in the earlier work (see J2, J3, J5, J10 in Appendix I Figure 6). Digital switching is controlled by the TelosB mote dependent on 3 modes: operation, calibration, and compensation.

3. Digital Potentiometer
   The manual potentiometer adjustment replaces digital potentiometer adjustment. This component is controlled by the TelosB mote. R, depicted above, is its variable resistance connected to one branch of the Wheatstone Bridge.

4. Flow Sensor
   This the thermal flow sensor detailed in Chapter 1.2. Its heater is powered by the Wheatstone Bridge and outputs ΔV (the voltage difference between the two thermopiles) to the differential amplifier in Flow Signal Amplifier.

5. TelosB Mote
   The mote is programmed with the master control program, written in NesC language. TelosB mote runs on TinyOS (see Appendix I for more details on TinyOS and NesC). The mote is programmed with two applications - one for normal operation and another for calibration testing. The latter has a faster sampling rate of the flow sensor.

6. HostPC (software applications)
   PC runs the Main GUI Application to visualize the data sent from the TelosB mote. This application reads the received data from the base-station and calculates an average value for the flow samples within a moving window, and then deduces the speed of the air flow and displays it. The Test Application is used to read the test measurement data during the calibration process. Subsequently, its data is processed by the Matlab Data Analysis Application to deduce the Air Flow Speed vs. Flow Values relation. A display will show the average speed of the flow (an example of 2.3 m/s is shown).

Power Supply Doubler, as the name suggests, doubles the supply voltage and powers up the Bridge Power Supply. The Bridge Power Supply is controlled by the Digital Quad Switch and the *Read signal* from the TelosB mote. Read signal is switched ON every second under normal operation, every 100 ms under calibration testing. Reference Voltage for Diff. Amp. provides a reference voltage of 1.2 V[4] to the Flow Signal Amplifier.

2.3 Hardware Design Hypothesis

Although shown separately for clarity, Amplifier Circuit, Digital Quad Switch, and Digital Potentiometer form a single unit. This unit connects to the TelosB, flow sensor, and the battery power supply; TelosB connects via 2 connectors; heater of the flow is powered by the circuit, and it samples the flow readings.

The circuit unit is responsible for 3 main tasks:

1. Autonomous balance of the Wheatstone Bridge.
   The Digital Potentiometer acts on the command signals received from the TelosB. In an algorithm specific manner, it raises or lowers its resistance and balances the lower half of the bridge.

2. Autonomous compensation of the Wheatstone Bridge for temperature.
   In order to sample the flow sensor, the *balanced* resistance value—as per above—of the Digital Potentiometer is raised by another ΔR (see Equation 9). Therefore, to balance the Wheatstone Bridge, a current should be passed through until the heater resistance raises itself by an amount equal to ΔR. This is Constant Temperature method (shown in the figure below).

[4]Only specific technical values are given in this Chapter. Details are covered in Chapter 3 and 4.

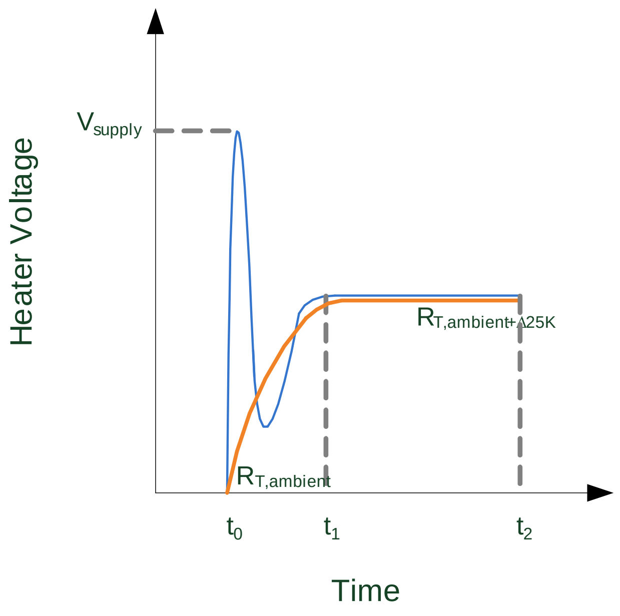


Figure 7: Flow measurement - Constant temperature Method

3. Sample the flow sensor periodically.
   This is the main, intended purpose of the circuit. Flow sampling is always preceded by the two tasks above, provided the balancing of bridge is deemed necessary according to a set criteria. This criteria is dependent on the ambient temperature (see *Criteria 1* in Chapter 3.1). Flow sensor is sampled during the last 10 ms of the t_1 to t_2 in the above figure, where it is most constant.

2.4 Software Design Hypothesis

The software application, programmed in to TelosB, is the main controller. Two different programs are used: one as the main program to sample the calibrated sensor; other is to help in testing, i.e. calibration process. The ins and outs of the softwear are described in length in Chapter 3.4. Therefore, following flow diagram is only a concise guide to help understand how the software controller works. Only two loops are visible; the other loops required to run the controller are nested within the latter two. Frequency of running some loops are determined on accessed necessity. Algorithm provides provision to change the such requirements as desired.

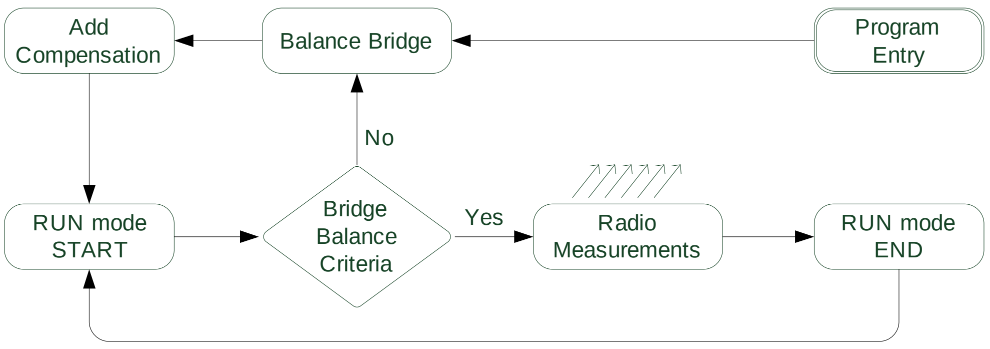


Figure 8: Software Controller

Upon entry to the program, initially, the bridge is balanced. Then compensation is added. Subsequent to these two steps, the system is ready to take flow measurements (and of other sensors mounted on TelosB mote). System operates on 3 main modes: operation (or run mode), calibration[5], and compensation. Program enters the RUN mode and check if the Bridge Balance Criteria is satisfied; if yes, it samples the sensors and radio them to the base-station; if no, it enters calibration mode and balances the bridge once more. Then the program marks the end of the sampling session and repeats the RUN mode.

2.5 Testing Hypothesis

The system underwent heavy testing to calibrate the sensor. Sensor calibration aims to achieve the relation as described in Chapter 2.1. The calibration process demands 5 requirements:

1. Air flows of different speeds ranging from 0 to 5 m/s[6].
2. Different ambient temperatures.
3. Means of achieving laminar air flow speeds.
4. Means of channeling the air over the sensor.
5. Means of achieving different temperatures.

Points above are discussed in detail in Chapter 4.

[5]Note that this calibration represents balancing of the bridge, different from the calibration of the flow sensor.
[6]In general, air flow speed within a container does not exceed 5 m/s

3 PROJECT DESIGN

This chapter discusses all inherent details of the circuit, applications, and controller programs briefed in the previous chapter.

3.1 Working Principle

Prior to discussing the working principle of the system, certain component blocks introduced in the hypothesis—and repeated in 3.1—need to be detailed. Wheatstone Bridge and Bridge Power Supply components are shown in the figure below.

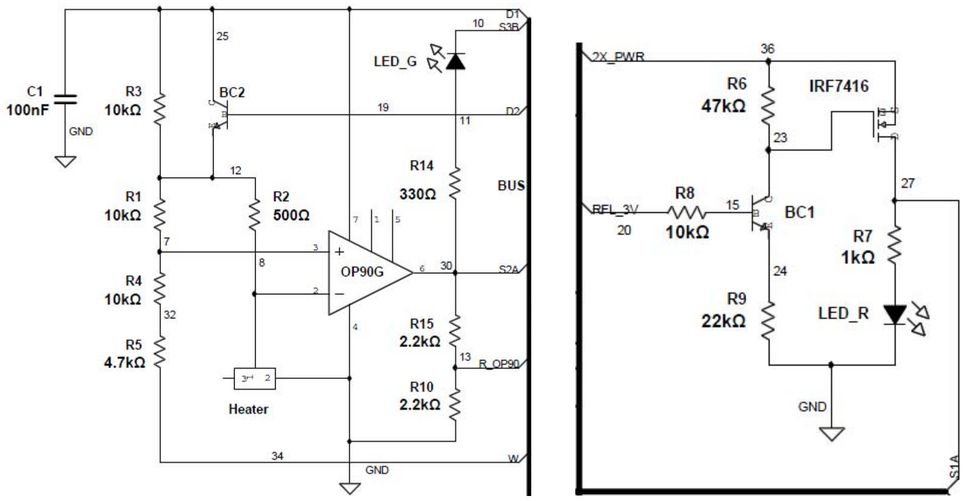


Figure 9: Wheatstone Bridge (Left) and Bridge Power Supply (Right) components

The line in ***Bold*** represents the bus. The inputs and outputs to the bus of the above figure are explained in the table below.

Signal	I/O	Description
D1	Common	Common pin of the S1 two-way switch of digital quad switch. Connects to either S1A or PWR (main power, 4 V)
D2	Common	Common pin of the S2 two-way switch of digital quad switch. Connects S2A.
S1A	O	Voltage doubler output is accessed via this pin. It is connected The Bridge Power Supply is power with the voltage doubler output.
S2A	O	Direct output of the OP90 (Pin 6).
S3B	O	S3B connects to GND to complete the circuit during balancing to light up the Green LED.
R_OP90	O	The output of the voltage divider connected the direct output of the OP90. Connects to TelosB U2 connector.
RFL_3V	I	The TelosB pin 7 signal (3V Read Signal).
2X_PWR	I	Voltage doubler output is connected to this pin.
W	NC	One end of the variable resistor in the Digital Potentiometer.

Table 1: Bus Signal Notation

See Figure 6 for following descriptions:

Power Supply Doubler	MAX1683
Reference Voltage for Diff. Amp.	LM385
Bridge Amplifier	OP90 within Wheatstone Bridge (above figure)
Digital Quad Switch	ADG_734
Digital Potentiometer	MAX5483
Flow Signal Amplifier	MAX4462H
Flow Sensor	Heater, TP1, and TP2 (disassembled)

Working Principle of the Circuit and TelosB

The following flow diagram illustrates the working principle of the circuit, coupled with controller signals from TelosB. This section is better referenced with the mini project circuit (Figure 6), the new circuit (Figure 6), and Chapters 2.2, 2.3 and 2.4.

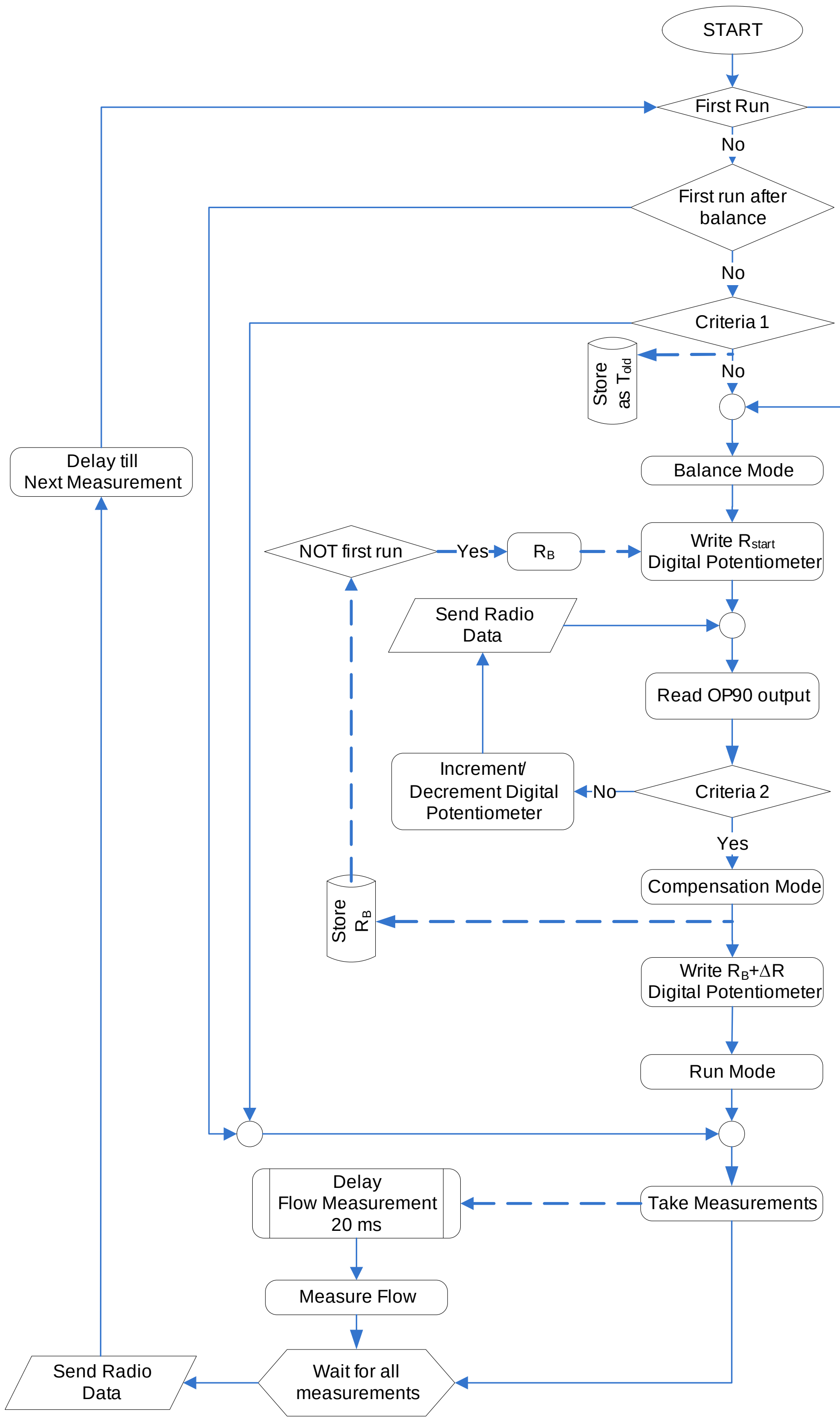


Figure 10: Working Principle

Note: dashed arrow lines indicate sub-processes, whereas the straight arrow lines indicate main processes. R_B is the resistance of the digital potentiometer at balance point.

When TelosB is booted, it checks a flag to ascertain if it is the first run. If yes, system goes in to *Balance Mode*. In Balance Mode, the digital potentiometer is written to with a predefined resistance value. It may or may not result in balancing the bridge. Therefore, OP90 output is checked to see if Criteria 2 is satisfied.

Criteria 2: OP90 must be within a predetermined range OR the swing counter (defined in Chapter 3.4) must reach its limit. The reason for having a bi-conditional criteria is due to indeterminable exact value of the heater resistance of the flow sensor at the time of performing a bridge balance and the high sensitivity of the bridge for each step increase of the digital potentiometer.

Subsequently, if criteria 2 is not satisfied the digital potentiometer is either increased or decreased. If the OP90 output is in the low range, a step increase results; if in high range, a step decrease results. This iterative loop is repeated till criteria 2 is satisfied. Then the system enters *Compensation Mode*. It adds $\triangle$R amount to the digital potentiometer, determined by Equation 9. Thereafter, the system enters *Run Mode* and takes measurements (temperature, humidity, and battery voltage). It also launches a subroutine (a background process) to delay the sampling of flow sensor by 20 ms. This is to ensure that the measurements are taken during the last 10 ms of the region t_1 to t_2 in Figure 2 (Note: t_0 to t_2 is 30 ms). When all measurements are completed (sampling duration defers vastly for different sensors: temperature/humidity approx. 200 ms, flow approx. 10 ms), all measurement data is sent via the radio. Subsequently, the system then waits a time determined by the master timer and repeats the loop. In the following loop it is checked for First Run flag, and then proceeds to Criteria 1.

Criteria 1: $(T_{old} - 1)^{o}C < T_{new} < (T_{old} + 1)^{o}C$ where T_{new} is the most recent (last) temperature measurement and T_{old} is one temperature measurement before the last.

Therefore, since T_{old} is unavailable, program proceeds to take measurements. In the following loop, the program enters criteria 1 once more and enters Balance Mode if criteria 1 is unsatisfied; or, it proceeds to take measurements. The program loops until powered down.

Working Principle of the GUI

On the host PC side, the data is read and displayed as see in the figure below. The program is written in LabVIEW.

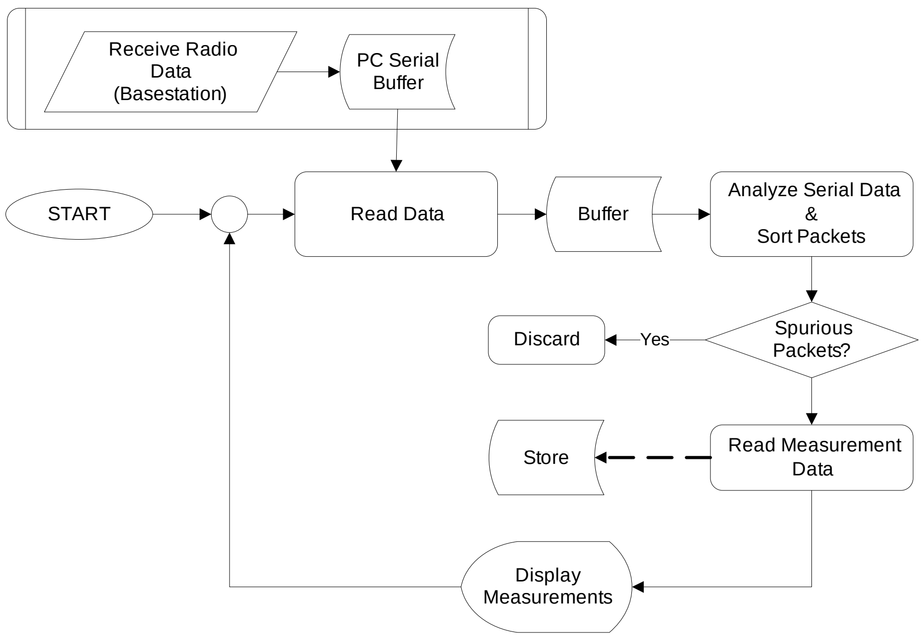


Figure 11: GUI Working Principle

The base-station data is read continuously as received from the TelosB mote. Subroutine component on top of the above graph shows the latter process. It is read serially and buffered in internal PC buffers (host controller USB buffers). Since buffers operate on FIFO concept, if the user GUI is not activated, internal buffers could overflow, yet retaining the most recent data in memory.

When the program is initiated, internally buffered data is read and buffered in program designated memory locations for program-related purposes. This is done as the program itself has no directives over the internal PC buffers for data control - only to read and write. Then the program buffered data is first checked for the packet frame indicators (0x7E) at the end and the beginning. Any serial data that do not comply with the latter is discarded as spurious. The rest of the data is sorted and disassembled—packet by packet—to read the information within. In the main application, temperature, humidity, battery voltage, sequence number of the packet, balance step of the digital potentiometer, etc. are displayed. The time is also recorded for analysis purposes. The read data is then displayed; thereafter, the program loops. The read loop illustrated above has a duration of 100 ms. Read loop resides within a much larger GUI program. The details of which will be covered in Chapter 3.4.

3.2 Hardware Design Implementation

3.2.1 Air Channel

The flow sensor measures the speed of air flow based on a stream of air passing just over and parallel to its surface. It is important for the air stream to be as laminar as possible. Otherwise, the resulting thermopile voltages may fluctuate, yielding spurious results in calibration.

Therefore, a housing is required to direct air to pass just over the surface, flow to be as laminar as possible, and width of flow not to be more than 1.5 mm. This housing is commonly know as a *channel*. The channel design was inspired by the channel used with flow sensors to measure fluid speed (see Figure 5 of [1]). The dimensions of certain characteristics of the new channel design are similar to that of its predecessor, e.g. inner channel and two smaller ones on either side. However, to accommodate new requirements, a redesign is required.

Basis for Channel Redesign

Some analysis is required on the old design in order to approach the redesign basis. Following diagram[7] represents the relation between Air Flow Speed vs. Thermopile Voltage Difference. It is of paramount importance to note the values of the axes, it is this fact the entire redesign is based on.

[7]These empirical data are from an experiment conducted in IMSAS in 2007.

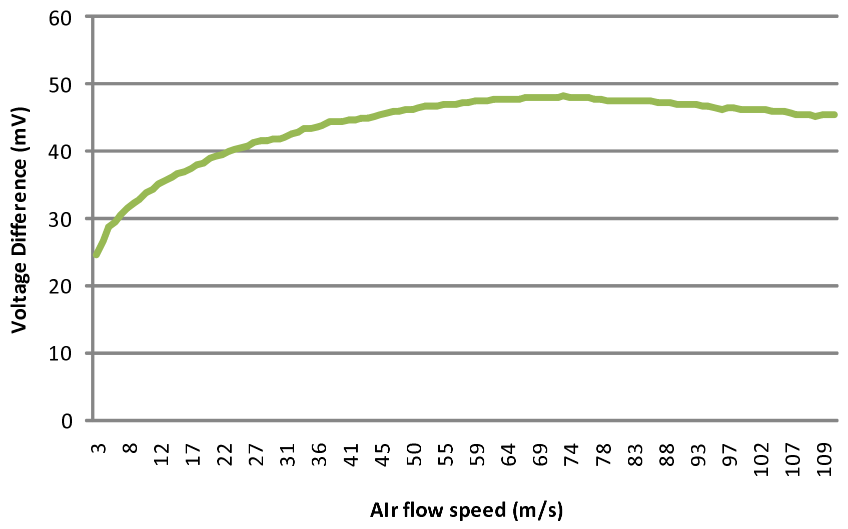


Figure 12: Air Flow Speed vs. Thermopile Voltage Difference

The dimensions of the channel used in the above experiment are 1.5 mm width and 1 mm height (cross section 1.5 mm^2). The air flow through the channel is measured with SLM (Standard Liters per Minute), which is converted to m/s using the cross sectional area. Data is obtained at 23.9°C for TS10_1 type flow sensor.

It is evident that reliable data for air flow speed starts approximately around 3 m/s. The voltage difference peaks approximately at 70 m/s. Therefore, it is possible to find a relation for this portion of the graph. However, for the purposes of determining air flow speed in a container, the above relation is unusable.

Therefore, a work-around is required. The idea is to step up the low air flow speeds (0 - 5 m/s) by a factor so that the resulting air flow speeds fall within the region identified above.

Continuity Equation in Fluid Dynamics

Continuity equation at steady state:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \qquad (2)$$

where ρ is the density of fluid and $\mathbf{u}$ the fluid velocity in vector form. For incompressible fluid, the above equation can be simplified as follows:

$$\nabla \cdot \mathbf{u} = 0 \qquad (3)$$

Therefore, basically, the mass continuity equation becomes a volume continuity equation. Assuming, the air passing over the flow sensor in this work is incompressible, the flow speeds can reach speeds in the region identified in Figure 3.2.1. It can be achieved as follows:

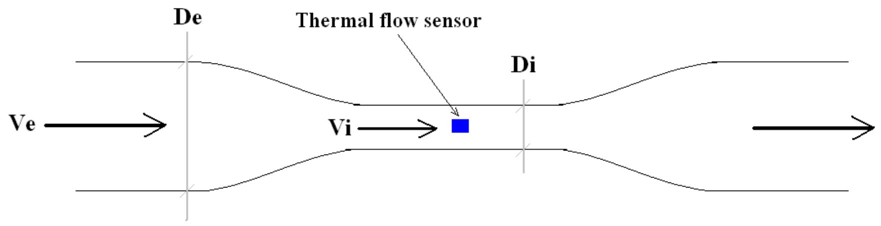


Figure 13: Two section air channel

Equation 3 becomes:

$$D_e^2 V_e = D_i^2 V_i \tag{4}$$

However, since air is compressible the above equation does not hold in above form. The sudden air pressure—in trying to squeeze in to a smaller diameter section—creates turbulence. If the air stream is applied long enough the air squeezes, overcoming the turbulence. Due to the created turbulence, the air density in the wider section (ρ_e) and the narrower section will change with time (t). These changing density functions are unknown and not within the scope of this thesis to determine them. However, using these time dependent density functions, it is possible to form an alternate mass continuity equation from Equation 4:

$$\rho_e\left(t\right) D_e^2 V_e \approx \rho_i\left(t\right) D_i^2 V_i \tag{5}$$

Although the ratio $\rho_e\left(t\right)/\rho_i\left(t\right)$ is indeterminable, it is reasonable to assume—provided the flow persists long enough—the flow speed of the air in the internal channel becomes higher than the external air speed. This hypothesis can only be proven by reverse deduction: if air flow speeds of 0 to 5 m/s yields a reasonable relation in testing, it is because the air flow speeds in the internal channel reaches the region recognized above in Figure 3.2.1.

Based on this assumption, a new two-section channel is designed. Due to manufacturing difficulties, a smooth curvature is not achievable between the two sections. The channel is designed with *Solid Edge*^{TM}. The channel is machined out of perspex.

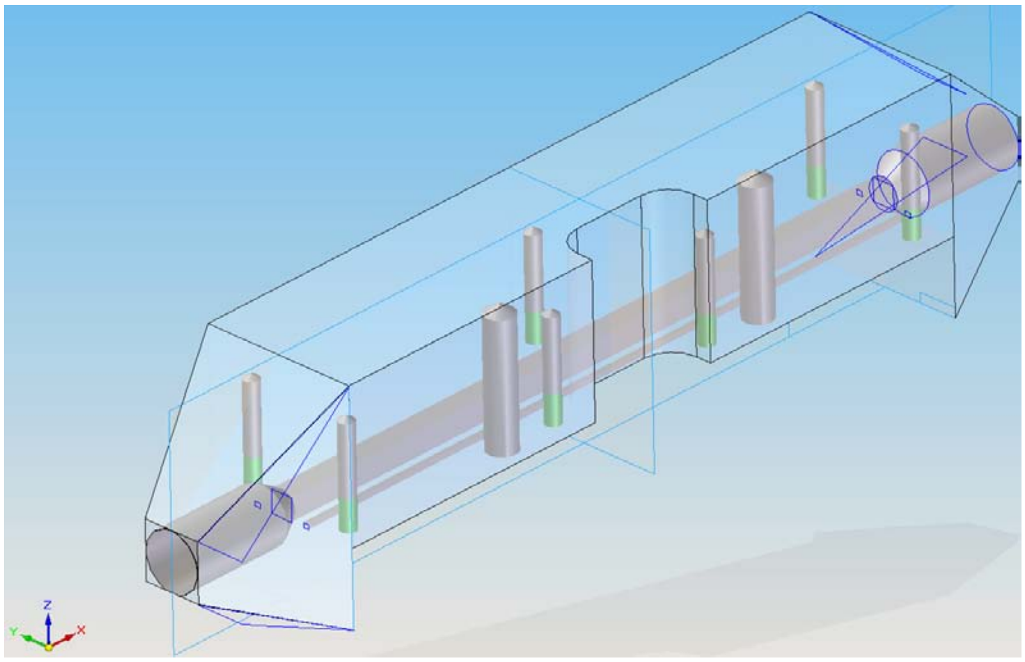


Figure 14: Solid Edge 3D Channel Design

The mechanical drawing can be found in Appendix III Figure 6.

Channel Area Difference of the Two-section Channel

The last figure shows a cylindrical, hollow channel on either ends of the channel with the internal, smaller channel in the middle. The reason to increase the air speed in the internal channel was established in the previous section. This is done by maintaining a large cross section in the outer channel, by changing the diameter of the hollow cylinder. However, the change of diameter is limited by the maintenance of laminar flow inside the internal chamber. Laminar flow is quantifiable with the Reynolds Number. The Reynolds Number equation is applied to the internal chamber to assess the maximum possible air flow in the internal channel, *assuming* the air is incompressible.

$$Re = \frac{VD}{\upsilon} \tag{6}$$

where Re is the Reynolds number, V the mean fluid velocity, D the characteristic dimension, and υ the kinematic viscosity. The characteristic dimension for a non-circular duct is calculated with $D = \frac{4A}{P}$, where A is the cross sectional area of the duct and P the wetted perimeter (Therefore, $D = 1.5\,mm$). Wetter perimeter of a rectangular duct as the one used in this design is 4 times the length of one side (the channel is a square type duct). The equation is rearranged to calculate the maximum mean velocity of air in the duct for $Re = 2000$. In fluid dynamics, it is a generally accepted fact that the flow is laminar for $Re < 2300$. However, leaving a safe margin, V_{max} is calculated for $Re = 2000$.

$$V_{max} = \frac{\upsilon Re}{D} = \frac{1.8 \times 10^{-5} m^2 s^{-1} \times 2000}{1.5 \times 10^{-3} m} = 24\,ms^{-1}$$

The considered maximum air flow speed in a container is 5 ms^{-1}. Therefore, this velocity should yield an increased maximum velocity of 24 ms^{-1}. Then, using continuity equation (Equation 4), the external diameter that satisfies the above criteria can be calculated.

$$D_e = \sqrt{\frac{D_i^2 V_{i,max}}{V_{e,max}}} = 3.29\,mm$$

where i and e means internal and external, respectively. The mechanical drawing of the channel in Appendix III shows a diameter of 3.85 mm. This is actually 3.80 mm, due to manufacturing limitations of the width of the drill bit. However, this is an acceptable diameter consistent with the above theory, because raising the Reynolds number to 2300—above the safe margin (used in the calculation)—and limiting the maximum external flow velocity to 4.5 ms^{-1} (5 ms^{-1} was not tested in this project), gives an external diameter of 3.71 mm, in the neighborhood of the actual diameter.

The above calculations are not directly applicable in this project as the air is compressible, and there is an unknown factor to be assessed (Equation 5). However, the assumption that the air is incompressible was used to arrive at an acceptable external diameter based on theory. Verification of the above is not within the scope of this project.

3.2.2 Digital Components Overview

The final circuit is depicted below in abstraction with all its inputs and outputs (except GND) for easy reference.

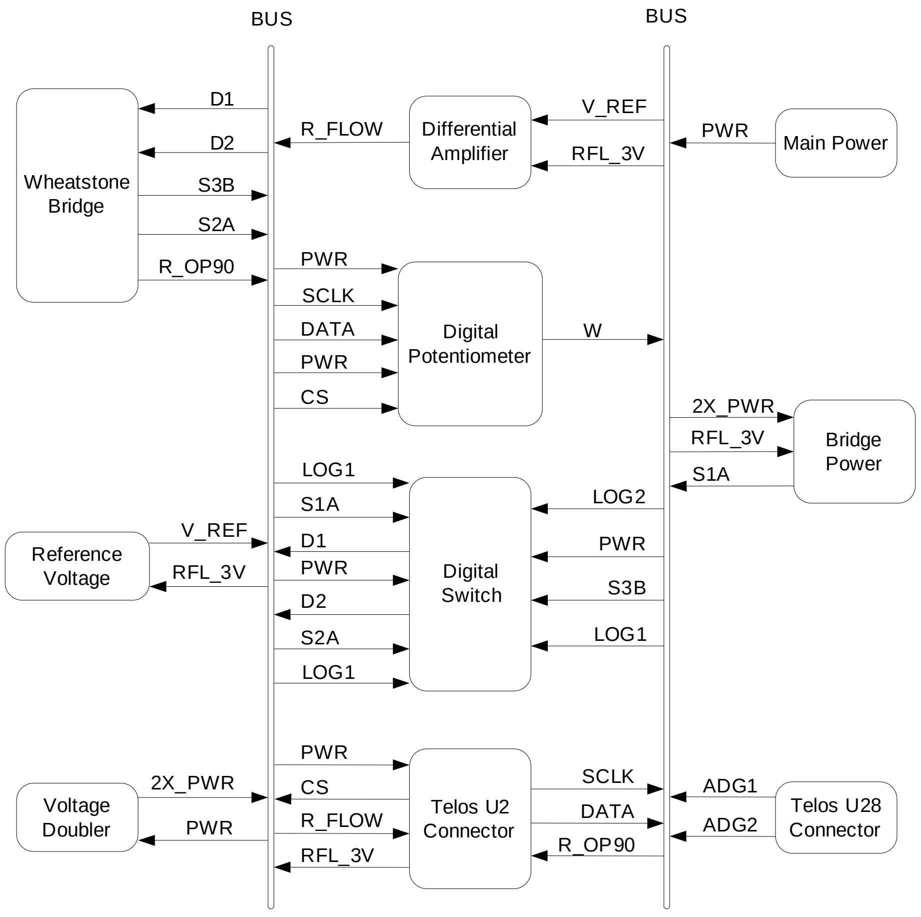


Figure 15: Overall circuit flow diagram with all inputs and outputs of the common bus

Digital Quad Switch - ADG734BRUZ This 20-pin digital device is in place for the 5 manual jumpers (J2, J3, J5, J10, and J14) operated in the previous design.

Pin configuration on ADG734BRUZ chip:

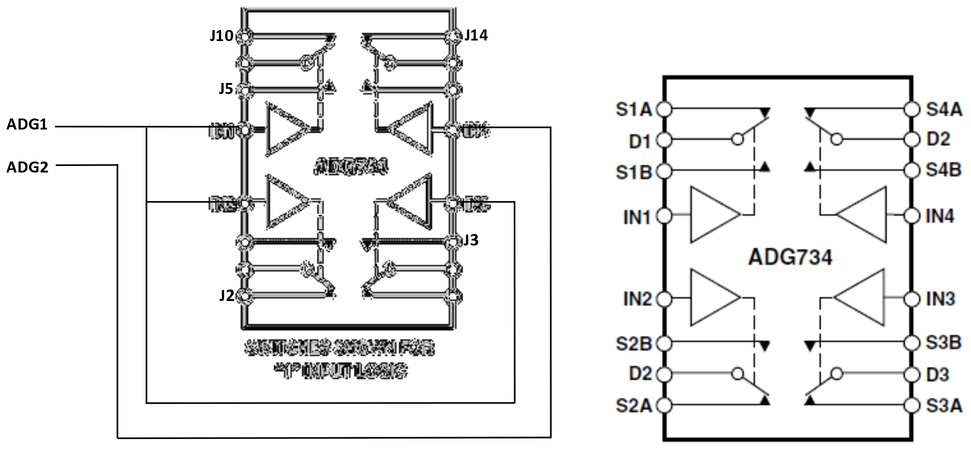


Figure 16: Pin configuration of ADG734 (left: showing manual jumpers, right: with original pin names)

Note ADG1 and ADG2 pins on the left. They are U28 (Figure 3.2.2) connector pins of the TelosB. They are used for logic simplification to reduce the number of pics required to automate the—previously manual—system. The digital switching is easy to operate: the pics marked above as IN1 to IN4 are logic inputs. These 4 dictate what switches are open and what are not. If the first switch is taken, for example, when IN1 is '1' the switch is closed between D1 and S1A; when IN1 is '0' D1 and S1B are closed [7].

The following table lists out the equivalence of the digital and manual switches in the old and new designs.

Manual Jumpers	**Digital Switches**	**Ref. Name**	**Description**
J2	D2 ↔ S2A	S1	Connects the feedback loop from OP90 output to the base pin of the transistor in the Wheatstone Bridge during flow measurement (Run Mode).
J3	GND ↔ S3B	S2	Completes the circuit to light up the Green LED in Balance Mode.
J5	D1 ↔ PWR	S3	Powers up the Wheatstone Bridge with main power (4 V) in Balance Mode.
J10	D1 ↔ S1A	S4	Powers up the Wheatstone Bridge with 8V (voltage doubler output) in Run Mode.
J14	Not used	S5	Not used.

Table 2: Manual and Digital Switching Equivalence

Note: J14 was used in the previous circuit design to isolate the wheatstone bridge in compensation mode. However, it is not required in the new design because compensation can be done without isolating the digital potentiometer. it is shown here only to retain the equivalence.

Pin requirements reduction on U2 and U28 extensions on TelosB Applying logic simplification for the digital pin requirements of the application (Figure 3.2.2) for the three modes on which the system operates. The following table deduces that three modes can easily be accessed by switching the ADG1. ADG2 pin is not required for branch isolation is not required with the digital potentiometer.

Operation/state	**S1**	**S2**	**S3**	**S4**	**S5**	**IN1**	**IN2**	**IN3**	**IN4**	**ADG1**	**ADG2**
Operation	1	0	0	1	1	1	1	1	1	1	1
Calibration (balance)	0	1	1	0	1	0	0	0	1	0	1
Compensation	0	1	1	0	0	0	0	0	0	0	0

Table 3: Logic simplification

TelosB Connectors Pin configuration on TelosB extensions U2 and U28 can be seen below.

Pin description	**U2 pin number**
V+	1
Digipot_CS	3
Read signal, 3V	5
CLOCK	6
Read air flow	7
DATA	8
V-	9
ReadOP90	10

Pin description	**U28 pin number**
ADG1	1
ADG2	2

Table 4: Pin configuration of U2 and U28 extensions

Pin configuration:

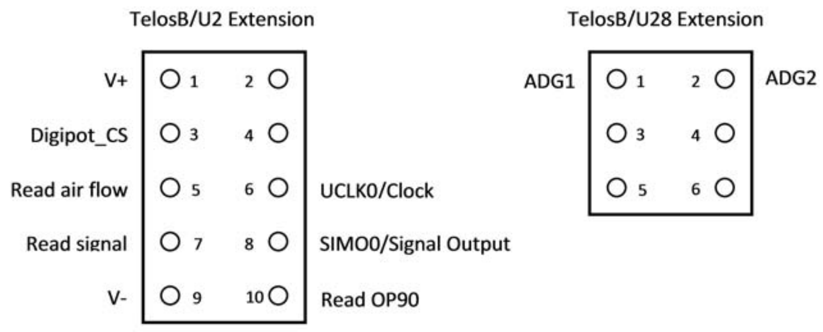
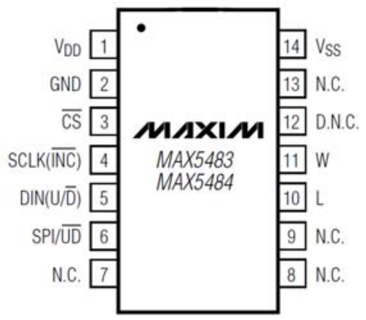
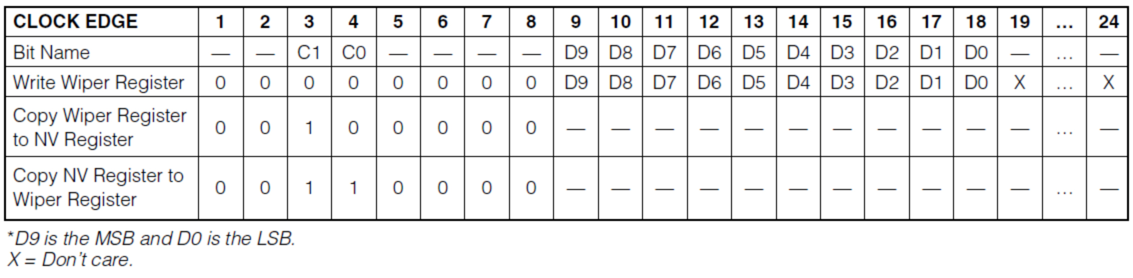


Figure 17: Pin configuration of TelosB extension U2 and U28

Digital Potentiometer - MAX5483 This is a 1024 tap 10 kΩ digital potentiometer with 10 Ω/step resolution. The resistance range is accurate for the actual resistance of the internal resistor from 70 to 10070 Ω. The pic configuration of the device is shown below.





CLOCK EDGE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...	24
Bit Name	—	—	C1	C0	—	—	—	—	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	—	...	—
Write Wiper Register	0	0	0	0	0	0	0	0	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	X	...	X
Copy Wiper Register to NV Register	0	0	1	0	0	0	0	0	—	—	—	—	—	—	—	—	—	—	—	...	—
Copy NV Register to Wiper Register	0	0	1	1	0	0	0	0	—	—	—	—	—	—	—	—	—	—	—	...	—

*D9 is the MSB and D0 is the LSB.
X = Don't care.

Figure 18: MAX5483 Pin Configuration (Top) and Command Decoding (Bottom)

D0 to D9 above are the 10 data bits, which can accommodate up to 1023 step values. C1 C0 are configured to be either a write or a read command.

The device is SPI (Serial Peripheral Interface) enabled. Therefore, TelosB platform can be effectively used with MAX5483, hence the reason for selection. It requires pin 3, 4, 5, and 6 to operate the SPI. It is not possible to read the resistance value (denoted by L and W). For this application, reading the resistance value is not required. Pin 6 is the SPI enable pin and is directly wired to the V_{DD}. Pin 3 $\overline{CS}$ has to be HIGH when writing to the SPI with Pin 4 (clock pin) and 5 (data pin). After writing the data to SPI, to latch the data to the internal memory of the chip, $\overline{CS}$ has to be turned LOW. The write command is 24 bit long and needs a little tweak in TinyOS to program seamlessly. TinyOS SPI write on TelosB can be done only 8 bits at a time. Therefore, the programming of TelosB contains 3 write cycles:

1. First command write cycle: line 18 3.2.2
2. First data write cycle: line 19
3. Second data write cycle: line 20

The code snippet of the *component* that *provides* the *interface* that achieves the above functionality is given below:

```
module DigitalPotentiometerC
{
    provides
    {
        interface DigitalPotentiometer;
    }
    uses
    {
        interface SpiByte as SPIbyte;
        interface GeneralIO as DigipotPinOnOff;
    }
}
implementation
{
    command void DigitalPotentiometer.WriteDigipot(uint16_t cmd, uint16_t data)
    {
        call DigipotPinOnOff.clr();
        call SPIbyte.write((uint8_t)cmd);
        call SPIbyte.write((uint8_t)((data & 0x3FC) >> 2));
        call SPIbyte.write((uint8_t)((data & 0x3) << 6));
        call DigipotPinOnOff.set();
    }
}
```

Each write cycle requires no time delay in between; however, after latching the written data to its memory, delay timer of 300 ms is provided in the main code. This is to ensure enough time to radio the data up to that point and to enable the user to see the progress. But, this feature is not required if visualization of the Balance Mode progress is not a primary requirement. Therefore, the timer can be reduced up to 20 ms (radio requires approx. 10 ms) and was tested separately and successfully.

The variable resistance is calculated by Equation 7 [8].

$$R_{WL}(D) = \frac{D}{1023} R_{W-L} + R_Z \tag{7}$$

where $R_{WL}(D)$ denotes the resistance at step value of D, R_{W-L} the total resistance, and R_Z the offset at step value 0.

MAX1683 and MAX4462H The details of these two chips can be found in [1].

3.2.3 Wheatstone Bridge

The circuit has an unequal symmetry to increase the step resolution when changing the digital potentiometer. In reference to Figure 3.1, left top branch of the bridge is R1 and left right the R2. The bottom left branch consists of R4, R5, and R_{W-L} (the variable digital potentiometer resistor); bottom right is connected to the heater of the flow sensor.

In order to discuss the step resolution, the voltage difference of pin 3 and pin 2 of the OP90 needs to be analyzed, defined as V_Δ. The nominal flow sensor resistance measured at room temperature is approximately 795 Ω for the sensor used in this application; this varies for different sensors. The *multiplication factor* of the bridge is defined as R1/R2. What is required is to see how V_Δ behaves near the balance point, i.e. when OP90 output is near zero, with one 10 Ω step value increase or decrease of the digital potentiometer in the bottom left branch. Reason for doing so is due to the high open-loop gain of the amplifier [9]. Therefore, the the behavior of V_Δ at balance point is of high importance in Balance

Mode. Assuming m for the multiplication factor and R_R the total resistance of the right branches at room temperature, mV value of $V_\triangle$ can be defined as follows (bridge is powered with a 4V battery).

$$V_\triangle = 4R_1 \left(\frac{1}{mR_R} - \frac{1}{mR_R + 10} \right) 10^3 \tag{8}$$

A test was conducted to determine the amplification of the OP90, and at balance for a 10 Ω step increase, the output of the OP90 stood at approx. 700 mV. That indicates a gain of approx. 1200. Using this value, the following table can be obtained for single 10 Ω step increase at balance point for different multiplication factors.

m	10	20	40	80	160
$V_\triangle$ (mV)	2.3833	0.5961	0.1490	0.0373	0.0093
Amplified Output (V)/step	2,86	0,72	0,18	0,04	0,01

Table 5: $V_\triangle$ for different multiplication factors

It is evident from the above table that higher the multiplication factor the better the resolution. Therefore, the detection of the balance point is more precise if a higher multiplication factor is chosen. However, higher multiplication factors lead to undesired noise at the OP90 output. This complicates the detection process and requires additional circuitry to remove the noise. Lower multiplication factors yield higher amplified outputs, yet again leading up to difficulty in detection due to very large amplification outputs. Therefore, a multiplication factor of 20 was selected for its reasonable amplification output and low noise characteristics.

3.2.4 Printed Circuit Board (PCB)

The PCB, shown below, is planned to be miniaturized. It is double layered and the silkscreen makes it easy to identify the components described in the previous chapter. Components are soldered only on top. PCB at this stage is expected to undergo further tests and may accommodate other components if further improvements are made.

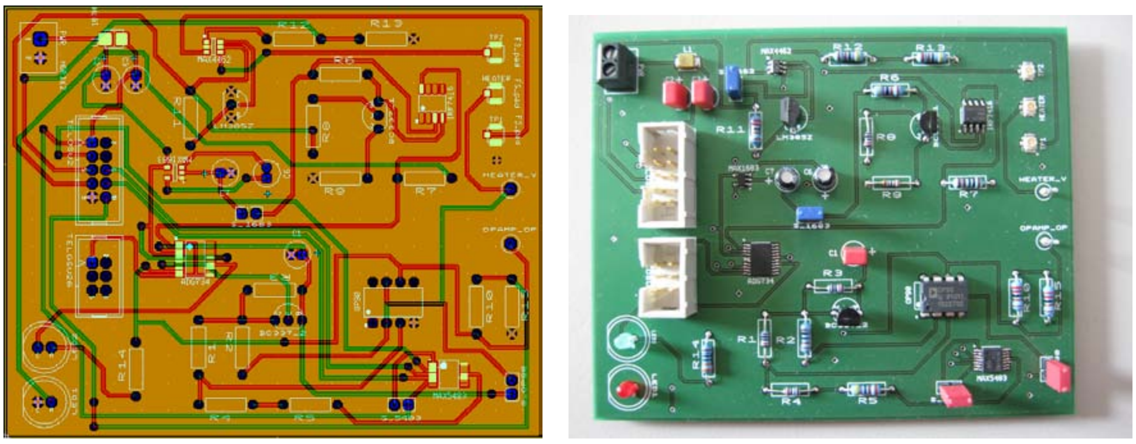


Figure 19: PCB (left: top and bottom views superimposed, right: actual circuit)

3.3 Automated Balance and Compensation Technique

The *Digital Potentiometer - MAX5483* section in Chapter 3.2 explains the write cycle of the digital potentiometer, which is the heart of the balance and compensation process. The target is to balance the bridge so that the bottom left branch of the bridge (see Figure 3.1) is equal to the bottom right branch, the heater; thereafter, an additional resistance ΔR is added to the digital potentiometer in Compensation Mode.

Bridge Balance Technique

The actual resistance of the heater at a given temperature is unknown. Therefore, it has to be deduced. The method of deduction is called the balance technique. The balance is achieved if Criteria 2 is satisfied. It was earlier introduced in Chapter 3.1 Figure 3.1. That criteria in detail is as depicted in the code snippet below as a *task*[8]:

```
 1 task void CalibrateBridge()
 2     {
 3         if(((1300 <= last_OP90) && (last_OP90 <= 1500))
 4                     || (swingCounter >= SWING_THRESHOLD))
 5         {
 6             last_balance = dataSPI;
 7             last_calibrated_temperature = last_temperature;
 8             mode = MODE_COMPENSATION;
 9             SetMode();
10             call Leds.led1Off();
11             call Leds.led2Off();
12             post AddDeltaR();
13         }
14         else
15         {
16             if(last_OP90 > 1500)
17             {
18                 highRange++;
19                 if((lowRange != 0) && (highRange != 0))
20                 {
21                 swingCounter++;
22                 }
23                 call Leds.led1Toggle();
24                 call DigitalPotentiometer.WriteDigipot(CMD, --dataSPI);
25                 call SPIWriteTimer.startOneShot(300);
26                 call ResourceSPI0.release();
27             }
28             else
29             {
30                 lowRange++;
31                 if((lowRange != 0) && (highRange != 0))
32                 {
33                     swingCounter++;
34                 }
35                 call Leds.led2Toggle();
36                 call DigitalPotentiometer.WriteDigipot(CMD, ++dataSPI);
37                 call SPIWriteTimer.startOneShot(300);
38                 call ResourceSPI0.release();
39             }
40         }
41         signal FlowControl.MonitorCalibrationProcess();
42     }
```

Note: Reference Appendix II and CD for the full documentation.

It can be seen on line 3 a variable called last_OP90: this is the output of the voltage divider connected to OP90 output (R_OP90 in Figure 3.1). It is read by pin 10 of the U2 extension of TelosB; therefore,

[8]In TinyOS task is a *background process* that can be posted.

this 12-bit AD conversion results in a value between 0 - 4095. The tests conducted with a manual, precise potentiometer affirmed that the balance point falls between 1300 and 1500 (i.e. *balance region*). However, as per the sensitivity analysis in Chapter 3.2.3, it is evident that the balance point can easily be missed dependent on the starting resistance value of the flow sensor. This in turn is dependent on the starting ambient temperature. Therefore, a single criteria is not sufficient, hence the second criteria *Swing Counter*.

Swing counter identifies two regions: *low range* for OP90 values less than 1300 and *high range* for values greater than 1500. Therefore, near the balance point, if the balance region is overshot or undershot, one of the low or high regions will be marked as arrived—*swingCounter* variable is incremented by one. Then on the next run—with the digital potentiometer increased or decreased accordingly—if the last_OP90 value overshoots the balance region again and reach the other region, *swingCounter* is incremented again. In general, from the tests conducted, the balance region is hardly reached when there is an ongoing *swing*. Therefore, a swing threshold of 5 (SWING_THRESHOLD) is declared; upon reaching 5 the bridge is declared balanced. The resulting error is marginal, a maximum of one step value of the digital potentiometer.

Aforesaid balance process has bottlenecks at reading the OP90 value and sending the updated variable data over the radio for observation and monitoring purposes. The total time taken for the balance process depends on how far the start value (START_CALIBRATE_VALUE, a constant provided in the header file *SensingData.h*) is from the actual resistance of the heater. In general, with one loop of the above code snippet taking less than 30ms, most of the balance processes completes within 0.5 s. In general, it is safe to limit the completion time of the balance process to less than 1 s.

Temperature Compensation Technique

Subsequent to the balance process, compensation is added. This is to make sure the flow sensor is sampled on Constant Temperature Basis. It is important to note the two code snippets—in-module function and a task—shown below:

```
uint16_t GetdR()
{
    if(last_calibrated_temperature >= 0x0F78)
        return 11;
    else if(last_calibrated_temperature >= 0x1040)
        return 10;
    else if(last_calibrated_temperature >= 0x1234)
        return 9;
    else if(last_calibrated_temperature >= 0x13C4)
        return 8;
    else if(last_calibrated_temperature >= 0x15B8)
        return 7;
    else if(last_calibrated_temperature >= 0x17AC)
        return 6;
    else if(last_calibrated_temperature >= 0x193C)
        return 5;
    else if((last_calibrated_temperature >= 0x1B30)
            && (last_calibrated_temperature <= 0x1C5C))
        return 4;
    else
        return 0x0FFF;
}


task void AddDeltaR()
{
    dR = GetdR();
    if(dR != 0x0FFF)
    {
        dataSPI += dR * (BRANCH_FACTOR / DIGIPOT_STEP);
        call DigitalPotentiometer.WriteDigipot(CMD, dataSPI);
        call ResourceSPI0.release();
        call CompensationDelayTimer.startOneShot(500);
        //signal FlowControl.MonitorCalibrationProcess();
    }
    else
        report_error();
}
```

It can be seen that the ΔR (dR in the top code snippet) is not calculated as it should shown below.

ΔR introduced above is as follows:

$$\Delta R = TCR_H . \Delta T . R_H \tag{9}$$

ΔR	Required rise of resistance (Ω)
TCR_H	Temperature Coefficient of Resistance (1/K)
ΔT	Temperature difference (25K)
R_H	Heater resistance at starting temperature (Ω)

The reason is to avoid floating point calculations, simplicity, and inability of the potentiometer to add decimal-point ΔR values (e.g. 7.4 Ω). The compensation values must always be multiplications of 10s for the minimum step value of the potentiometer is 10 Ω. Therefore, to solve this problem, calculations using the above equation are done separately and dR is deduced based on the last recorded temperature, at which the Balance Mode was reached (Criteria 1 in Chapter 3.1). If it is on compulsory Balance Mode at boot-up, the temperature is measured before going in to Balance Mode. The table below shows the results of latter calculation. This calculation has to be performed for each flow sensor individually. The dR values are rounded to nearest unit integers. Therefore, there are temperature regions with the same dR value, which is perfectly substantial for the actual change of heater resistance over few degrees Celsius is low.

Temperature Region	**Temperature (°C)**	**Temperature (0x)**	**dR** (Ω)	**dR (rounded)** (Ω)
d	16	15B8	7.4	7
d	17	161C	7.2	7
d	18	1680	6.9	7
d	19	16E4	6.7	7
d	20	1748	6.5	7
c	21	17AC	6.3	6
c	22	1810	6.1	6
c	23	1874	5.9	6
c	24	18D8	5.6	6
b	25	193B	5.4	5
b	26	199F	5.2	5
b	27	1A03	5	5
b	28	1A67	4.8	5
b	29	1ACB	4.6	5
a	30	1B2F	4.3	4
a	31	1B93	4.1	4
a	32	1BF7	3.9	4
a	33	1C5B	3.7	4

Table 6: Temperature Compensation dR Calculation

The above table only shows 4 temperature regions for which the calibration tests were conducted. Testing the flow sensor in different temperature regions is dependent on the availability of such ambient temperature regions, hence its high dependence on weather.

The dR value deduced above has to be further compensated for the bridge multiplication factor. Increase of dR amount on the left branch is not equivalent to increase of dR on the heater side. The reason is the unequal left and right branches of the bridge. Therefore, dR needs to be multiplied by a factor determined by $\frac{BRANCH\,FACTOR}{DIGIPOT\,STEP}$ (i.e. 2).

3.4 Software Design Implementation

Three separate software applications were used for the project.

1. TelosB control and data collection application
   This is by far the most important application. This program governs the entire embedded system including all sensors and the data communication with the host PC applications.

2. GUI application
   There are different forms of this application[9]: one for displaying data for the calibrated flow sensor and the other to record data during the calibration testing phase. LabVIEW was used as the programming language. The main application sorts the error-free packets received by the base-station and displays the data graphically. The test application is used for calibration data display and record. It records the data and writes them to text files to be analyzed.

[9]The core application module was written by Dirk Hentschel of IMSAS for a different application. The code is reused for this app with many changes.

3. Calibration data analysis application
   The data text files recorded by the LabVIEW test application is read by a Matlab program, and some graphs (see Chapter 4) are produced with the raw data. Then some manual inspection and calculations are performed, after which those data is fed again to another Matlab application to yield further more graphs and to perform curve fitting in order to obtain the relations indicated by Equation 1 (Chapter 2.1).

TelosB control and data collection application is detailed in this chapter. Other two applications are briefed in Chapter 4 where is it more pertinent. Most important and relevant code and supplementary information are provided in the appendices, and the rest of the data is provided with the CD.

3.4.1 TelosB Application

The application programmed in to TelosB is two-fold: one for the sampling of the calibrated flow sensor, the other to sample test data during the calibration of the sensor. This section discusses the general application for sampling the calibrated sensor. The other application—with minor changes to the latter—is discussed briefly in Chapter 4.

The programming is done with NesC language (Appendix I) on TinyOS as the operating system. The code is compiled along with TinyOS together as one program and uploaded in to TelosB. The reason for the usage of NesC is its high efficiency for WSN related applications. The program uses integer calculations only. The TelosB application is explained below in detail with code snippets and flowcharts, except for balancing and compensation processes which were detailed in the earlier sections.

The whole program code can be divided mainly in to 3 parts:

1. Main configuration (ControllerAppC.nc)
2. Main module (ControllerC.nc); Sub modules (FlowSensorControlC.nc, DigitalPotentiometerC.nc)
3. Interfaces (SensorControl.nc, FlowControl.nc. DigitalPotentiometer.nc)

Refer to Appendix II Table 6 for the full file list and Appendix II section TelosB Program Code for their program code.

ControllerAppC contains all the *wiring* required to connect all interfaces and user defined *modules* to the system modules used in the program. The main module houses the *booted* event[10]. Booted event is the first event signaled after boot-up of TelosB, and it initiates the program with some system related parameter initialization. Program execution is shifted from one file to another according to the ControllerC file.

The entire program can be described in two loops: first loop compulsorily balances the bridge and compensates for target temperature; the second loop is the normal program execution to sample the flow and other sensors. These two loops are explained below with flowcharts and noteworthy code snippets.

First Loop

Note: T, H, B, and F stands for Temperature, Humidity, Battery Voltage, and Flow.

[10] *event* is a TinyOS specific type word.

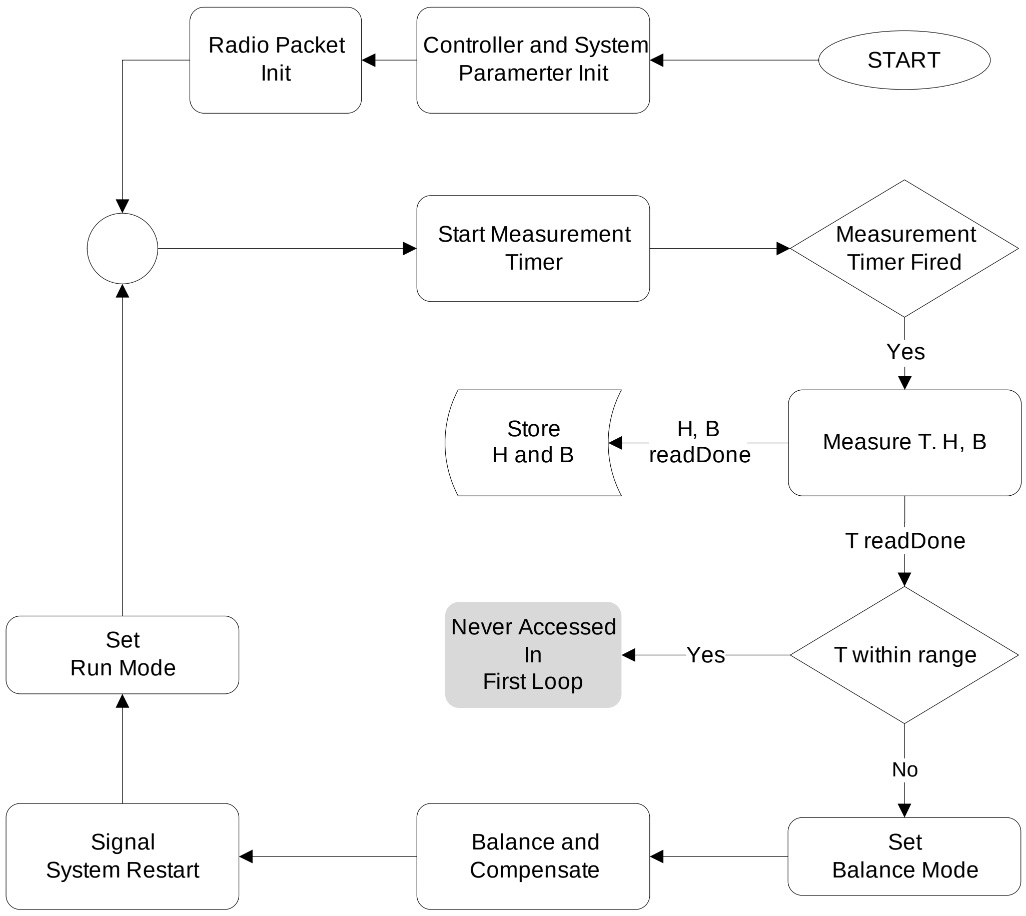


Figure 20: First Execution Loop of TelosB Program

After boot-up and system parameter initialization, radio packet header is also initiated; afterwards, radio send frequency is set. In this application, radio sends the collected data every second. Measurement Timer is then *called* to keep track of how long the measurements take, which is required to set the Send Timer correctly to allow enough time for the sensors to read data and ensure constant data relay frequency. Subsequent to *firing* of the measurement timer, the first measurements of T, H, and taken.

The program then signals the *readDone* events once the measuring is complete, and the data is saved. However, inside the T readDone event, measured temperature is checked to see if within a predetermined range (Criteria 1, Chapter 3.1). In the first loop, the variable last_calibrated_temperature is set to 0xFF9B (i.e. 0xFFFF - 0x64) to force the *false* condition of the if loop. The code snippet of the T readDone event and MeasuremetTimer.fired event is given below.

```
event void ReadTemp.readDone(error_t result, uint16_t data)
{
    if(result != SUCCESS)
    {
        data = 0xffff;
        report_error();
        //if error keep the old value
    }
    else
    {
        if(((last_calibrated_temperature - 0x64) < data)
                && (data < (last_calibrated_temperature + 0x64)))
        {
            last_temperature = data;
            post DelayFlowMeasurement();
        }
        else
        {
            last_temperature = data;
            signal SensorControl.SystemRecalibration();
        }
    }
}


event void MeasurementTimer.fired()
    {
        if(measurementMode)
        {
            call SensorControl.OneMeasurement();
            measurementTimeStamp = call MeasurementTimer.getNow();
            startSendTimer();
        }
        else
        {
            call SensorControl.PrepareBuffer(measurementTimeStamp);
            startMeasurementTimer();
        }
    }
```

This forces the system to enter Balance Mode; therefore, the Balance Mode is set and the bridge is balanced and compensated. Subsequently, the Run Mode is set and the measurement timer is restarted. Within this first loop, after the balancing is completed, last_calibrated_temperature variable is assigned with the measured temperature value. Therefore, first loop concludes here and then starts the normal operation of sampling the flow sensor.

Second Loop

When the measurement timer is fired for the second time (Note: this particular timer is used for two purposes.), it can go in to two sub-modes (see code snippet above): start measurements mode and radio measurements mode. On the first following the first loop described above, the system samples all sensors once more. In the T readDone event—as seen in the code snippet above—flow delay timer is called. This is to ensure 30 ms long time period for flow measurement, from setting the *flow enable pin* HIGH (pin 7, U2 of TelosB) to setting the same pin LOW after the measurement is complete (in flow readDone event). This flow delay timer is shown with a dashed line in the figure below. Program execution does not wait for the readDone events and proceeds to the next line in the program execution stack: Start Send Timer. This function (not a Timer) calls a measurement timer of 350 ms. This is to allow enough time for the temperature and humidity sensors to take measurements; on average they take approx. 250 - 300 ms cumulatively. Since there are no execution tasks left on the stack thereafter, all sampling of sensors are completed before the called timer is fired. After its firing, since the flag that determines the sub-mode is negated in the previous run, it goes in to radio mode. This prompts preparation of the radio buffer and sending of the measurements data and some other data (see lines 176 - 208 in FlowSensorControl.nc in Appendix II).

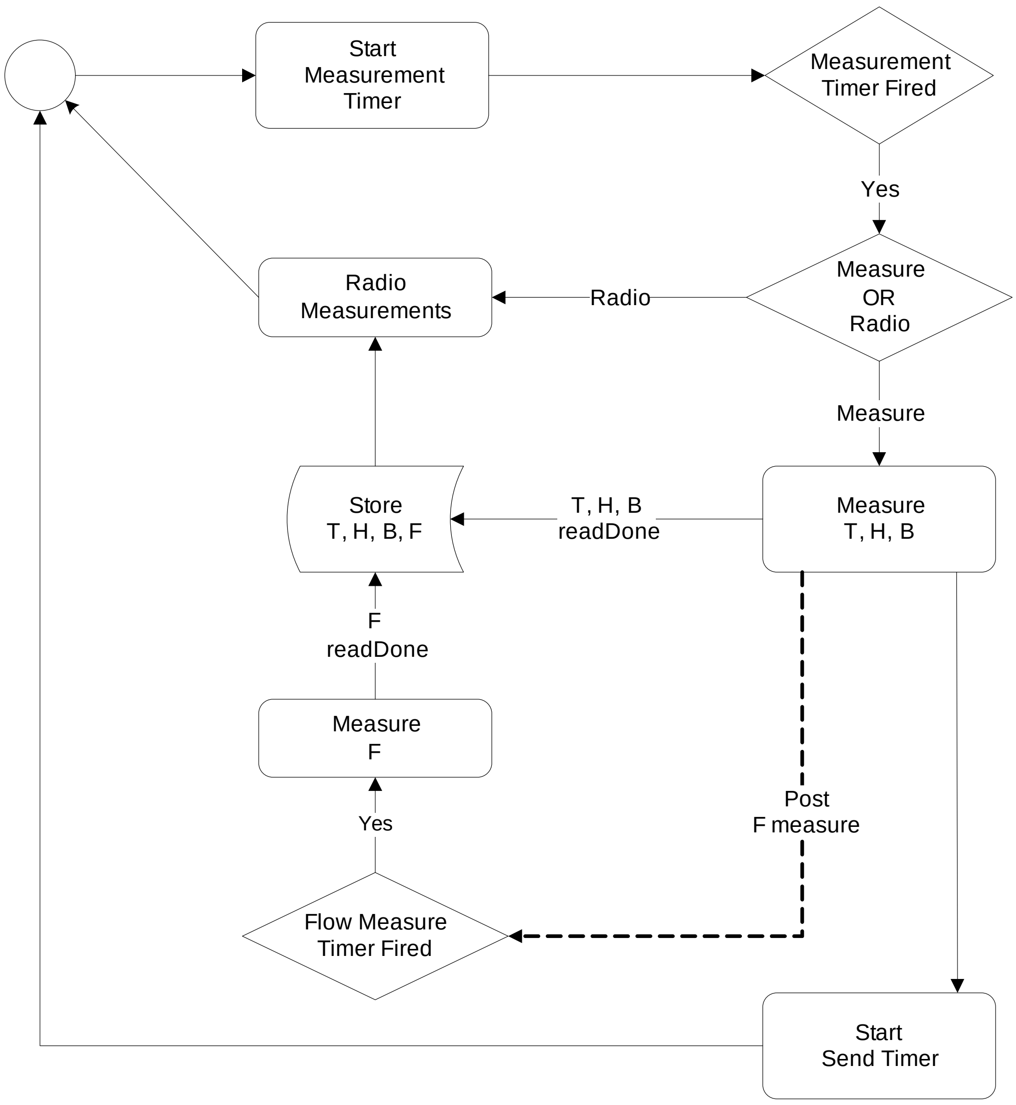


Figure 21: Second Execution Loop of TelosB program

The second loop is infinitely reiterated as long as Criteria 1 is met. If not, system balances and compensates autonomously before taking any more flow samples.

4 TESTING

The flow sensor cannot be used for absolute measurement of air flow because it is uncalibrated. Calibration essentially—in this case—means one-to-one mapping of voltage difference of the two thermopiles and air flow speed. This relation was introduced and mathematically represented in Equation 1. The voltage difference is measured with the AD converter of the TelosB. Therefore, the latter equation is rewritten with the measured AD conversion value, simply named $\overline{F}$. Note here that the used parameter for flow is the average flow reading. This is further explained in Chapter 4.3.

$$\overline{F} = f^{i}\left(\overline{\nu}\right) \tag{10}$$

This chapter explains *how* relation f is deduced, but the actual relation is given in the next chapter. Therefore, testing adopted a simple technique to pass a stream of air over the sensor area and measure the voltage difference, that too under different temperature regions explained in the previous chapter.

Note the 5 requirements listed in Chapter 2.5.

4.1 Testbed

At the time of testing, the testbed contained prototype circuit board, in every way similar to the final circuit PCB shown in this report. The testbed contains the toy car, flow sensor, prototype circuit boards, TelosB sensor board, 4 V battery, protective box for the boards, ON/OFF switch, and a USB cable to program the TelosB mote. Following figure illustrates the testbed.

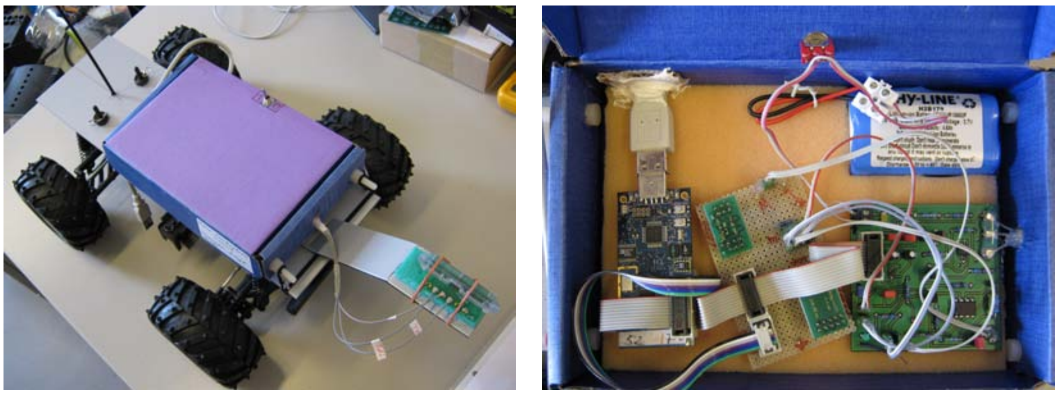


Figure 22: Testbed

4.2 Test Methodology

Air flow speeds Calibration of the flow sensor for this application is limited in scope. The reason being the air flow speeds present in an actual refrigerated container transporting goods are between 0 and 5 m/s. Therefore, some means of varying the air flow speed was required. This came in the form of a radio-controlled toy car. Instead of streaming the air at the sensor attached to a immovable stable base, sensor is moved at different speeds in stagnant air which remains at 0 m/s. The car features a speed controller, wheel alignment, approx. 20 m (indoor) radio range, an electric motor, and a 7.2 V heavy duty, rechargeable battery. The car is capable of achieving a maximum speed of 4.5 m/s, but maintaining lower speeds around 1 m/s and less proved to be difficult. Maintaining the mid-speeds, on the other hand, is easy. But, it is not possible to repeat the same speed twice one after the other; however, such precise repetitions are not required to calibrate the sensor. What is required is to produce different speeds of the said range as much as possible, not specifically in an ascending or descending order.

Ambient temperature The other most important factor in testing is the ambient temperature under which the testing is performed. The testing was performed indoors. Table 3.3 lists 4 temperature regions. During the period the testing was carried out[11], it was not possible to find temperatures less than 16°C (indoor).

Laminar air flow If the sensor is calibrated under laminar flow, it would result in the best calibration relation achievable. However, it is difficult to achieve laminar flow due to a variety of reasons: internal channel turbulence, manufacturing errors in the channel, drafts in test sites, car vibrations, car handling errors, etc. Therefore, the best possible conditions were selected under testing. Testing was always carried out indoors, after making sure that there are no drafts across and along the test trip at car-height level. Therefore, as can be seen in previous section, the channel end meets the air at 0 m/s. That avoids certain amount of turbulence, which otherwise would be present. Channel design is also carefully designed to make sure the best possible laminar flow. It is detailed in Chapter 3.2.1.

Test Methodology

Testing requires two persons: one to operate the the radio controlled car; and, another to record the time. The figure below illustrates the test setup. Different test locations were used to perform the tests. The choice of test location depended on the available ambient temperature, drafts, and a straight stretch of level ground.

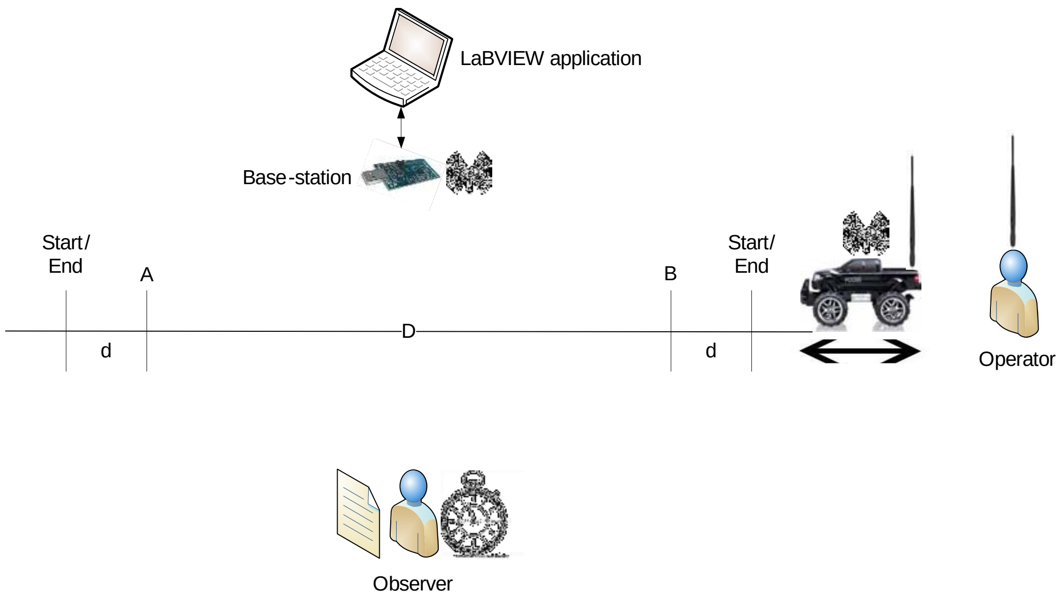


Figure 23: Test Strip

The test equipment setup is organized as follows: The car houses the electronics and the TelosB mote. It has two antennas: one for the car remote and the other the antenna of the TelosB mote. Operator controls the car, and it is run back and fourth between the 3 Start/End signs. The observer records the time the car takes from A to B and B to A depending on the driving direction. The car continuously transmits the flow measurements to the base-station. The base-station is connected to the laptop, which runs a LabVIEW Test program. It records all the raw data received, including some other data, such as system time, last balance step of the digital potentiometer (number of step values required to balance the bridge), compensation value (dR), OP90 divider output, etc. Note two different sets of two antennas signifying the difference between the mote-to-mote communication and the car remote.

[11]During the month from 5.7.2010 to 5.8.2010.

The stretch of level ground was set at 12 m (i.e. D, the distance between A and B. Some tests had 15 m distance). The distance d, between Start/End and A/B, is the *acceleration region*; it is about 1.5 - 2 m in length. The car starts at Start/End and accelerates for a distance of d. This, however, is hypothetically correct. Because the car is controlled manually, the acceleration region may fall short or longer. However, test observation and experience proved d to be a reasonable distance. But, when trying high speeds ($>$ 3.5 m/s), the distance d is doubled to allow enough time to achieve the desired speed.

The operator of the car remote maintains the speed at A/B constant for the full distance of D. The observer records the time taken for the car to travel from A to B or A to B. A to B or B to A is defined as one *test run.* The sensor is calibrated only on one direction of the channel. It is, of course, possible to measure the flow by rotating the channel horizontally by 180 degrees. However, due to time restrictions, calibration is only done for one direction. The battery of the car lasts for up to 45 minutes. Therefore, as many tests as possible were conducted within this time duration.

The car continuously samples the flow sensor, at one measurement each 100 ms, and radios immediately. Therefore, the send (radio) frequency is equal to the sampling frequency. At D = 15 m, $\overline{\nu} = 3$ m/s, for example, the program samples the flow sensor 50 times. For lower speeds it is higher, and for higher speeds the count is lower.

The base-station receives the data sent by the TelosB onboard the car, and the LabVIEW Test program (with minor changes to the application stated in 3.4) writes the received data to a text file (syntax: *fm_ <date>_ test<number>.txt*). The text file data is then read with a Matlab program and undergoes some manual manipulation, which is discussed in the next section. The manipulated is written on to another text file. This text file is read by another Matlab application and a graph is produced between the Air Flow Speed and Flow Output (ADC reading).

4.3 Test Results Extraction and Manipulation

As explained the previous section, the data transmitted by the car is recorded by the base-station and written to disk by the LabVIEW Test application. Altogether, 9 complete tests were completed. The text files of raw data are included in the CD. In total, 116,200 samples were received and recorded, however radio packet counter of the TelosB program recorded a much higher packet transmission number—the difference being the lost packets, amounted to about 15%.

The following code snippet shows the Matlab code (DataAnalyzer.m) that reads the raw data file. *Read-Measurements* function is simply the Matlab *textread* function. It prompts for user input to read a desired file, and it categorizes and produces different matrices belonging to different measurements, e.g. temperature, flow, heater balance, etc. Eventually, it produces a graph between Flow Value and Received Packet Number. Flow Values are extracted from FLOW matrix, and the Received Packet Number here is the same as FLOW matrix index.

```
filename = input('Enter file name: ', 's');
[DATE SYSTIME MOTEID, TIME, TEMPERATURE, SNUMBER, OP90,...
...MODE, FLOW, HEATER] = ReadMeasurements(filename);
plot(FLOW, 'r');
xlabel('Received packet number','FontSize',14);
ylabel('Flow Value','FontSize',14);
```

An example—test 7—of such a graph is shown below for analysis purposes.

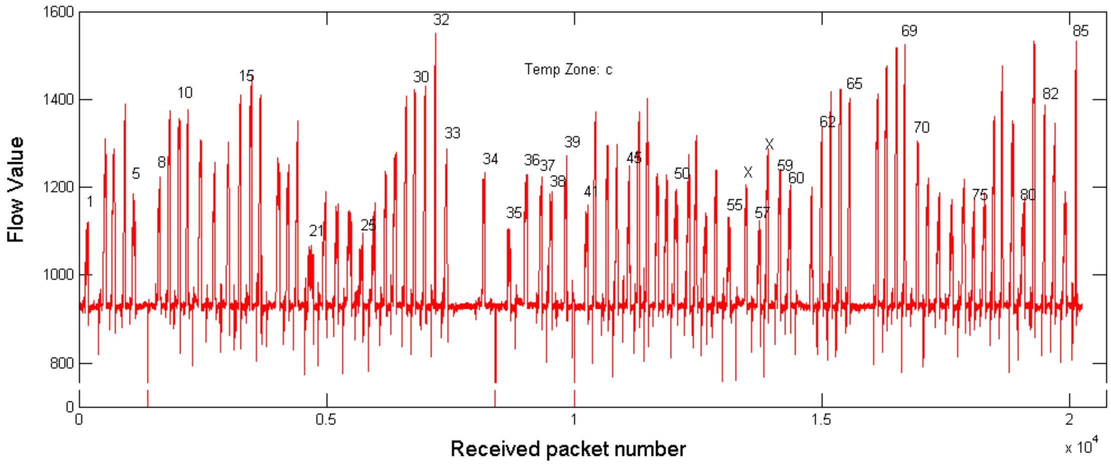


Figure 24: Air Flow Speed vs. Flow (ADC value), Temperature Region 'c'

Each peak shown on the graph belongs to an individual test run, from which the average flow speed is calculated. The Xs shown are the rejected or refused test runs due to various reasons: presence of drafts, high speed variations during test run, and erroneous timing. The downward peaks are also shown. These are when the system was manually reset for re-calibration (re-balancing and re-compensation). The temperature is taken few times during the entire experiment with a thermometer. There are no temperature measurements taken by the TelosB during the test run except in the start, where it goes in to a compulsory balance and compensation process.

The Main TelosB Application is slightly changed to adapt to a test program. There is a User Button on TelosB, which the application uses to start the application and for any other re-calibrations thereafter. See the code snippet of ControllerC.nc, changed as follows, for this purpose. Only the additional events, different from the main TelosB program, are shown. Notice line 5 which throws the event in line 11 when the User Button is pressed. This sets the Balance Mode, and it eventually leads to compensation, too.

```
 1  event void Boot.booted()
 2  {
 3      InitSystem();
 4      InitControllerParameters();
 5      call Notify.enable();
 6      call SensorControl.StartSensorControl();
 7  }
 8  .
 9  .
10  .
11  event void Notify.notify(button_state_t state)
12  {
13      if ( state == BUTTON_PRESSED )
14      {
15          call MeasurementTimer.stop();
16          call FlowControl.SetCalibrationMode();
17          call FlowControl.BeginCalibration();
18      }
19      else if ( state == BUTTON_RELEASED )
20      {
21          // do nothing
22      }
23  }
```

The calculation of the average speed is a manual, labor-intensive process. Average speed is calculated by visual inspection; to explain this visual inspection process it requires to zoom in to one peak (test run 18 of Figure 4.3) shown in the above figure.

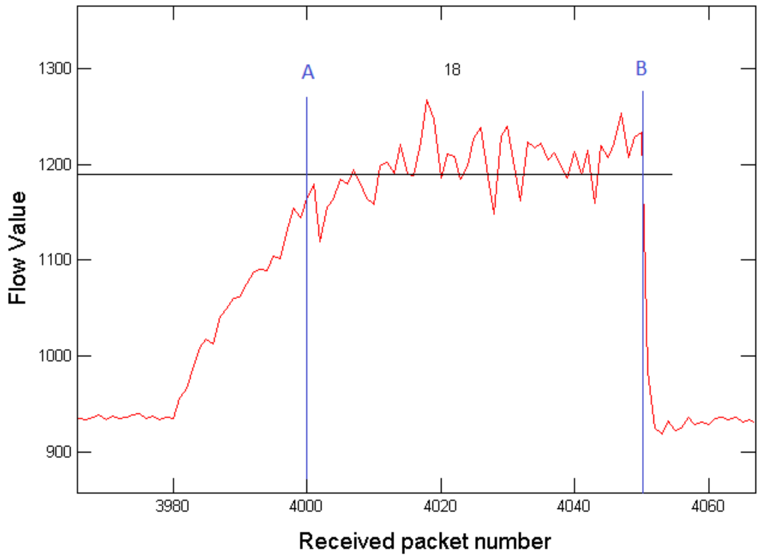


Figure 25: An individual test run

Time taken to complete the test run for the above curve ($t_{18,AB}$) 5.75 s; average speed $\overline{v}_{18}$ of the car 2.1 m/s.

The acceleration region is clearly visible at the beginning of the curve. Though with ups and downs, the curve continues to maintain an average flow value ($\overline{F}$) till the end. The sharp drop at the end of the curve is due to the operator engaging breaks at, say, B (see Figure 4.2). Therefore, the sharp drop (B) clearly marks the end of test run; now the beginning of the curve at which the timing starts has to be figured out. The distance d is approximately 1.5 - 2 m; therefore, 15 to 20 measurements are recorded during this time. By visual inspection, counting 15 - 20 measurements along the X-axis, it is possible to note the beginning of timing (A). Thereafter, using *ginput* Matlab tool, A and B are marked and $\overline{F}$ is calculated. In the above figure $\overline{F}$ is 1190. An obvious error is shown in the graph; that is the slight acceleration still present after point A, when timing starts. This is an unavoidable error depending on maintaining constant speed throughout the test run. Therefore, the process is susceptible to unavoidable human error. A new method by which human errors can be avoided is presented in the next section (Chapter 4.4) as a solution.

The average speeds of each test run is recorded this way and written to another text file using the following Matlab code snippet:

```
[x,y] = ginput(2);
fprintf(fid, '%5.1f\n', mean(FLOW(ceil(x(1)):floor(x(2)))))
```

These command lines are repeated for each test run. *fid* refers to the file ID of the text file the data is written to. *x(1)* and *x(2)* are the indexes of FLOW matrix (where flow values are read in to) refer to A and B of the particular test run, respectively. The index values of the FLOW matrix must be integers; therefore, *ceil* and *floor* functions are used.

The text file produced is named, for example, *st_<date>_test<number>_<temp region>.txt*. This enables the next Matlab program (See Appendix II *Matlab Program*) to plot the Average Speed vs. Average Flow for the tested temperature region—c, in this case.

The described process is repeated for all tests, and the aforementioned Matlab program plots all data in one graph.

Curve Fitting and Error Analysis

The visual inspection of the curves obtained points in the direction of non-linearity. Therefore, a second order polynomial—for a better fit—is considered for the relation f^i in Equation 10.[12]

$$f(\nu_i) = a_2\nu_i^2 + a_1\nu_i + a_0 \tag{11}$$

where a_0, a_1, a_2 are the coefficients of the polynomial, and $f(\nu_i)$ the predicted flow value. F_i values are the actual measured, average flow values. The calculation of these coefficients are done for temperature zones a, b, c, and d. Curve fitting uses the Least Squares method, which minimizes the error of the proposed curve with respect to the coefficients.

The error of the curve can be written down as follows:

$$error = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (F_i - f(\nu_i))^2 = \sum_{i=1}^{n} \left(F_i - \left(a_2\nu_i^2 + a_1\nu_i + a_0\right)\right)^2 \tag{12}$$

where n is the number of data points for a particular temperature zone. Now taking partial derivatives of the error with respect to the coefficients to nullify the error (i.e. $error = 0$)

$$\frac{\partial error}{\partial a_2} = -2\sum_{i=1}^{n} \nu^2 \left(F_i - \left(a_2\nu_i^2 + a_1\nu_i + a_0\right)\right) = 0 \implies a_2\sum \nu_i^4 + a_1\sum \nu_i^3 + a_0\sum \nu_i^2 = \sum \nu_i^2 F_i$$

$$\frac{\partial error}{\partial a_1} = -2\sum_{i=1}^{n} \nu \left(F_i - \left(a_2\nu_i^2 + a_1\nu_i + a_0\right)\right) = 0 \implies a_2\sum \nu_i^3 + a_1\sum \nu_i^2 + a_0\sum \nu_i = \sum \nu_i F_i$$

$$\frac{\partial error}{\partial a_0} = -2\sum_{i=1}^{n} \left(F_i - \left(a_2\nu_i^2 + a_1\nu_i + a_0\right)\right) = 0 \implies a_2\sum \nu_i^2 + a_1\sum \nu_i + a_0 n = \sum F_i$$

Now the above equations are solved for a_0, a_1, a_2 in matrix form.

$$\begin{bmatrix} \sum \nu_i^4 & \sum \nu_i^3 & \sum \nu_i^2 \\ \sum \nu_i^3 & \sum \nu_i^2 & \sum \nu_i \\ \sum \nu_i^2 & \sum \nu_i^1 & n \end{bmatrix} \begin{bmatrix} a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} \sum \nu_i^2 F_i \\ \sum \nu_i F_i \\ \sum F_i \end{bmatrix} \implies VA = F \implies A = V^{-1}F \tag{13}$$

Therefore, A can be solved, and with Equation 12 error can be calculated. Matlab provides useful functions such as ***polyfit*** and ***polyval*** for the calculation of latter. A manual calculation using the above equations and the Matlab functions yielded the same results. Therefore, for the analysis of curve fitting in the next chapter those Matlab functions are used[10].

4.4 Proposal for an Advanced Testbed

The calibration test methodology conducted on the thermal flow sensor presented several errors, mainly human errors which are avoidable. Therefore, with the experience obtained from the testing described in this report, a proposal for a better testbed is proposed for future development. It takes in to account the following error-prone aspects and provides solutions.

[12] For simplicity, $\overline{F}$ and $\overline{\nu}$ is considered here without the *average sign.*

1. Vibration reduction of the car.
2. Guarantee constant speed for the entire duration of the test run between A and B (i.e. Timing limits).
3. Automate timing without human intervention.

A toy train system—with manually installed automatic timers—on guide rails is proposed as the moving unit in place of the car used. This solution can be categorized in to several sections based on the following:

1. Track type—straight or circular
   Straight track type would be much similar to the tests conducted in this project. But, it requires the train to stop at either end of the track and restart the next test run in opposite direction. Whether it is done automatically or not, it is unnecessary additional work.
   A circular guide rail makes the latter task unnecessary. It enables the toy train to circle the track at different speeds without stopping till the end of the experiment. It is a much faster means of sampling the flow sensor at varying speeds.
2. Speed control—remote controlled or TelosB-controlled
   The same method of using a remote controlled car can be used with the straight track type (with a toy train, but not a car). The speed can be measured by installing a optical sensor onboard the train and two light sources at A and B.
   It is possible get more creative with the circular method: a full self-contained system can be developed with speed control, detection, timing, and flow measurement—all within one system.
3. Timing
   For the straight track type, timers can be activated when the light sensor is triggered (by the light sources). This time duration can be measured without a problem by TelosB timers.
   For the circular track type, along with the speed control method used for the train, only a speed confirmation method is required. This can be achieved with an onboard optical sensor and only one light source.

It is clear from above that a train on a circular guide rail is much an efficient system that a straight track system to simulate air flow speeds. The following model figure attempts to model this hypothetical testbed.

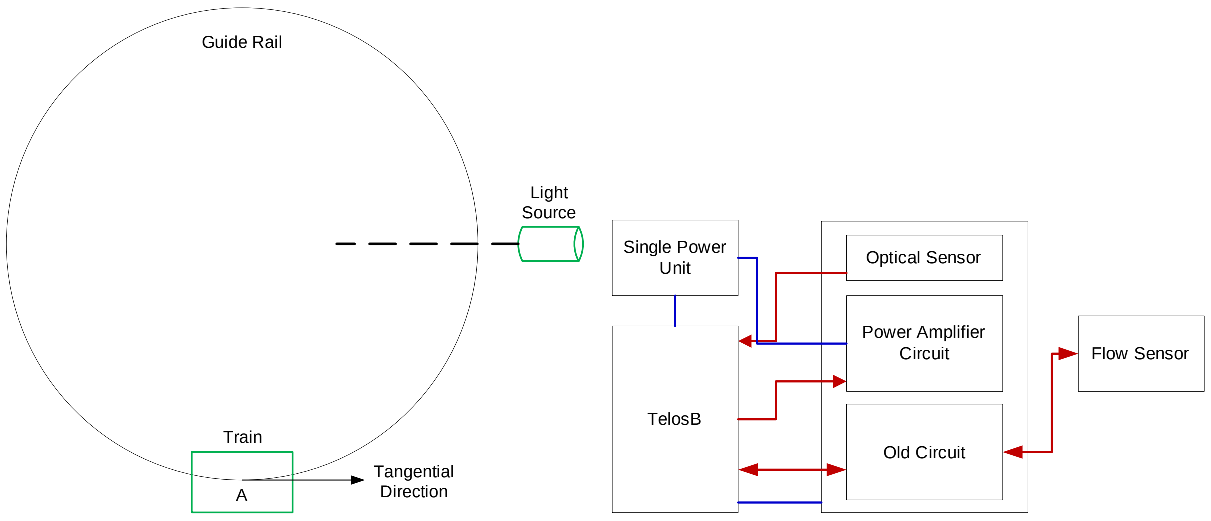


Figure 26: Proposed testbed

The noticeable changes to the circuitry of the previous system are the optical sensor and a power amplifier:

1. The optical sensor senses the light source each time the train completes one circle. This enables the TelosB to activate one of its timers and calculate the time taken per circle; thereby, it is possible to deduce the speed as the circumference of the guide rail can be precisely measured.

2. Power amplifier is to setup a speed control system for the DC motor of the train. PWM is a more efficient way of controlling speed of a DC motor. However, PWM implementation is quite straightforward in TelosB platform. But, by simply using the DAC, it is possible to vary the voltage of a pin on one of the extension slots of the TelosB. Therefore, using a non-inverting power amplifier circuit, the input voltage to the DC motor can be changed and—most importantly—kept *constant* for a given period of time.

The placement of the flow sensor also holds high importance. Note the point A of the above figure, located on the guide rail. The opening end of the sensor channel has to coincide with this point and the alignment of the channel length has to be along the tangential direction marked at point A. To avoid any turbulence caused by the train and its onboard equipment, the flow sensor must be mounted few inches on top of the train.

Another important detail of this system is that the flow samples can be saved in to the internal Flash Memory of the TelosB. Therefore, radio complications–such as missing packets and delays—can be avoided. The data can be retrieved later through a flash download program using a USB cable. Considering all of above, a small algorithm is presented for better understanding. Naturally, any real implementation of this system will result in a more advanced program, especially because this is a *self-contained* program.

```
//Proposed algorithm for an advanced testbed
START MAIN
Initiate system
Measure temperature
Balance and Compensate
START do while (Loop Timer <= 20 minutes) (say)
    Loop timer (T)
    Set DC motor speed OR Increase Speed
        //based on multiples of interrupt counter
        Set DAC to lowest value OR Increment
    Wait for Light Sensor interrupt
        interrupt counter++
        Get interrupt time (t)
        Get duration (delta_t)
        Set RUN mode for  flow measurements
        if (not started)
            Start flow measurements
        // Measure temperature every 5 minutes (say)
        Period temperature measurements
        // data = flow, delta_t, session number...
        // ...,temperature, balance point, etc.
        write data to flash
END do while
Stop flow measurements
END MAIN
```

The shown algorithm is to familiarize the reader with only a rough idea of the implementation of the proposed system. After system start, it sample the temperature sensor of the onboard TelosB; it then proceeds to balance and compensate the system, getting the system ready to sample the flow sensor. It then starts a *do while* loop for 20 minutes (say). Within this loop the system checks for the interrupt generated by the light sensor and marks that time. Upon receiving the next interrupt from the same sensor, it calculates *delta_t.* On reception of the first interrupt it starts a continuous flow measurement cycle, unhindered by the 20 minute loop. Upon the reception of each light sensor interrupt, the delta_t is measure and all the data is saved to the flash memory. After 20 minutes, the do while loop ends, and subsequently the flow measurements are also stopped. That concludes one experiment.

This system can run indefinitely as long as the battery holds. It can be programmed to stop automatically when the Main Battery Unit's voltage drops below a predetermined limit. Therefore, all that requires is to download the data and analyze them. Since the speed remains absolutely constant, and the fact that there are no start/end to distinguish, the train speed and the flow measurements are in perfect correlation. This helps to avoid the human errors stated in the beginning of this section.

5 RESULTS and ANALYSIS

The results obtained by testing can be categorized as below:

1. The raw measurement data from the TelosB
   These data files are stored in files of type *fm_ <date>_ test<number>.txt.* Most important data in these files are the flow values, radio packet sequence number, temperature, and dR values.
2. The averaged flow values and air flow speeds
   These data files are stored in files of type *st_ <date>_ test<number>_ <temp region>.txt* and *speed timing <date> test<number>.xls.*

The above data is used to plot Flow Values vs. Received Packet Number (from 1) and Average Flow Value vs. Average Air Flow Speed (from 2). The latter plot assumes the Least Squares method to find the plot equation (relation f^i). Therefore, an error calculation is also produced. Analysis of the results are included along with the results' plots.

5.1 Flow Value Plots for Different Temperature Regions

Ten experiments were conducted in total. Ninth experiment had to be abandoned; therefore, it is not accounted below.

Experiment Number	Date	Temperature at start (^{o}C)	Temperature variation (^{o}C)	No. of test runs	Accepted test runs
1	05/07/10	26.4	< 0.5	21	21
2	06/07/10	25.1	< 0.5	30	30
3	10/07/10	28.2	< 0.5	30	27
4	10/07/10	32.0	< 0.5	34	32
5	10/07/10	32.1	< 0.5	35	33
6	14/07/10	31.5	< 0.5	39	37
7	15/07/10	22.9	< 0.5	85	80
8	21/07/10	29.0	< 0.5	45	39
10	05/08/10	18.5	< 0.5	80	74

Table 7: The conducted experiments

The above experiments are plotted below. Each peak represents a test a run. The 'X's shown on top of some peaks are the rejected test runs, and the other numbers represent the numbers of successful test runs for the particular test. Following 4 graphs represent the 4 temperature regions under consideration.

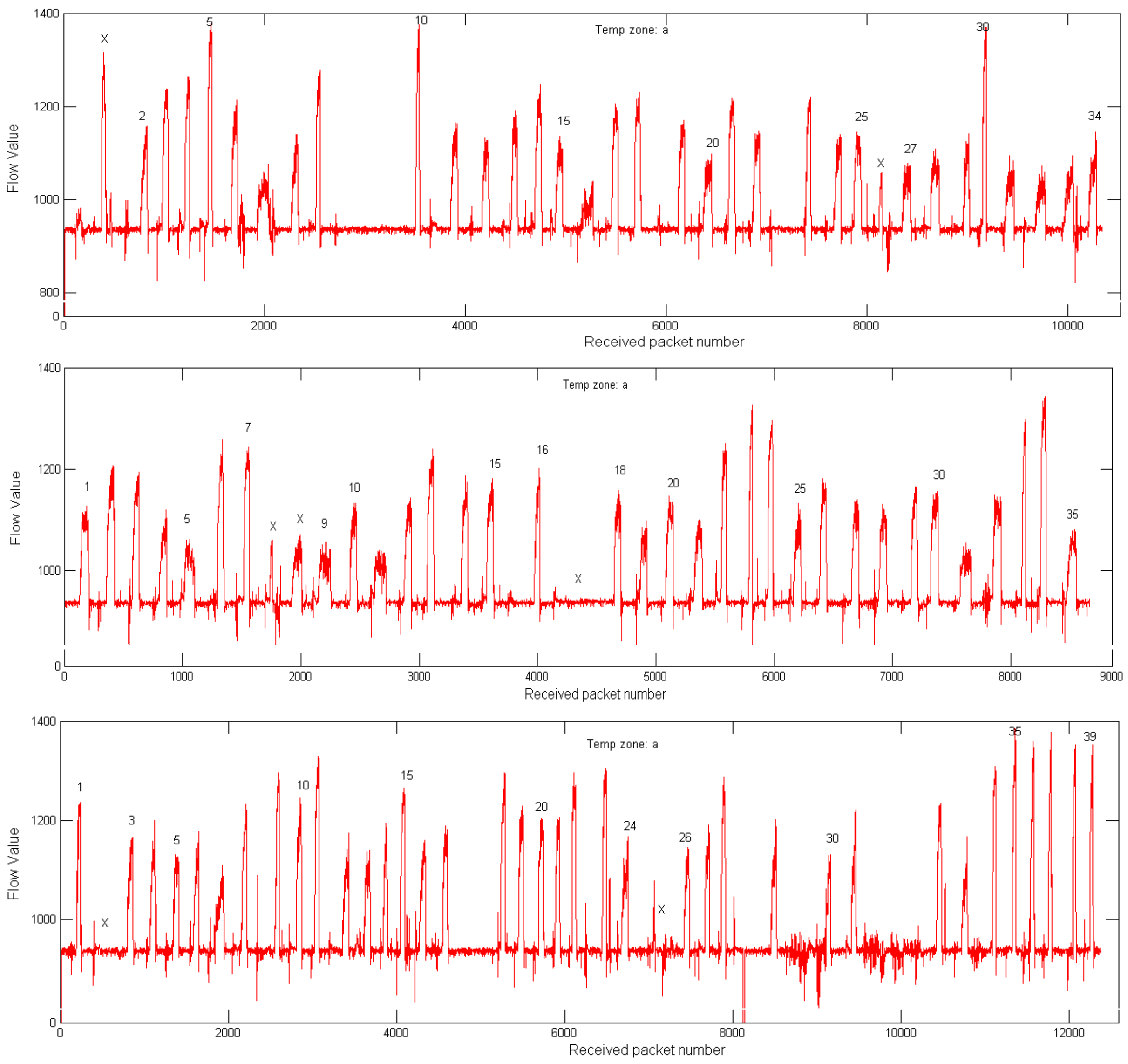


Figure 27: Flow Value vs. Received Packet Number for Temp Zone **a** (Top: test4; Middle: test5; Bottom: test6)

The above plot is for temperature zone **a**: $30 \le Temperature < 33$. The number of total test runs for this region amounts to 102.

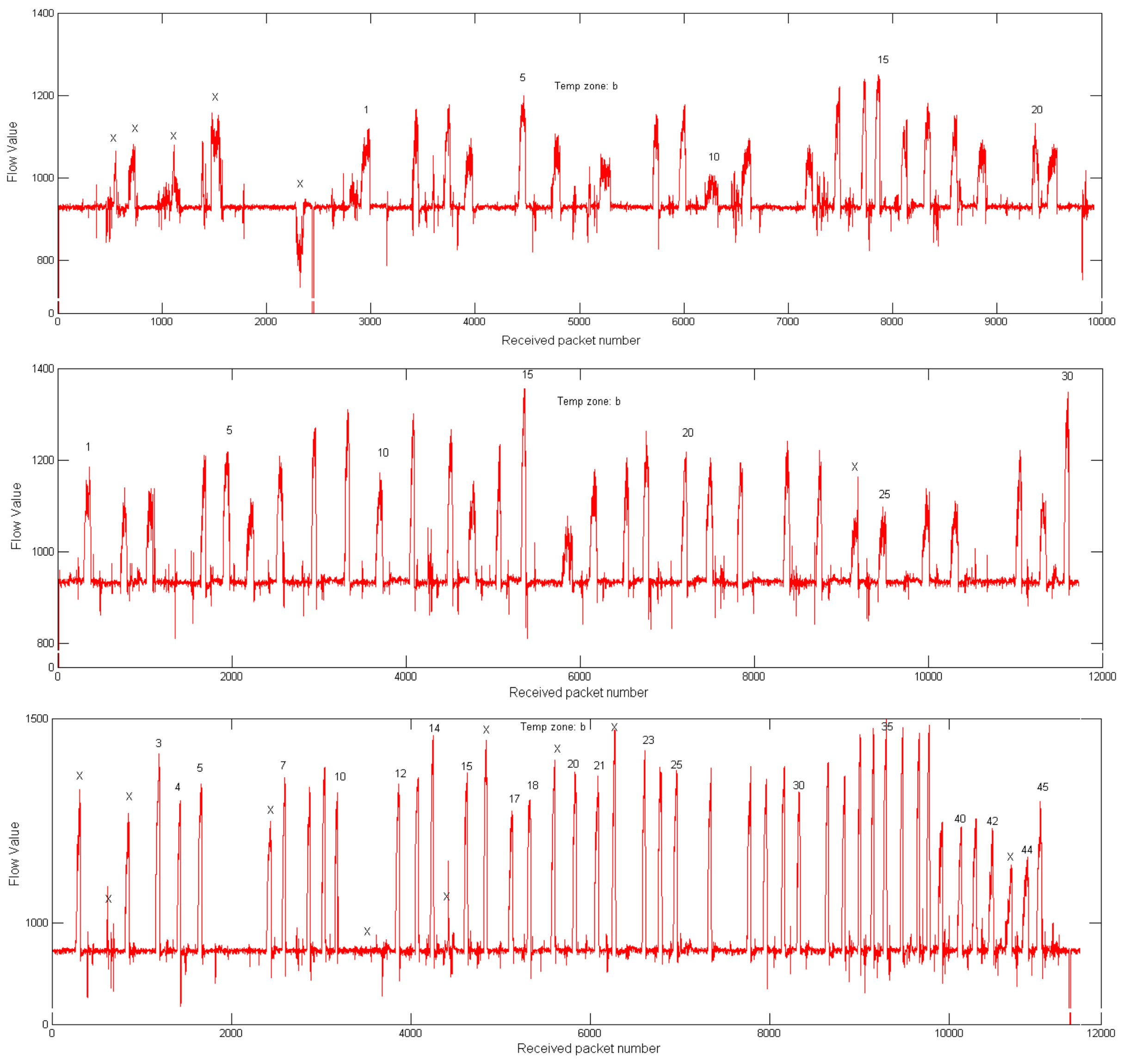


Figure 28: Flow Value vs. Received Packet Number for Temp Zone **b** (Top: test1; Middle: test8; Bottom: test2)

The above plot is for temperature zone **b**: $25 \leq Temperature < 30$. The number of total test runs for this region amounts to 115.

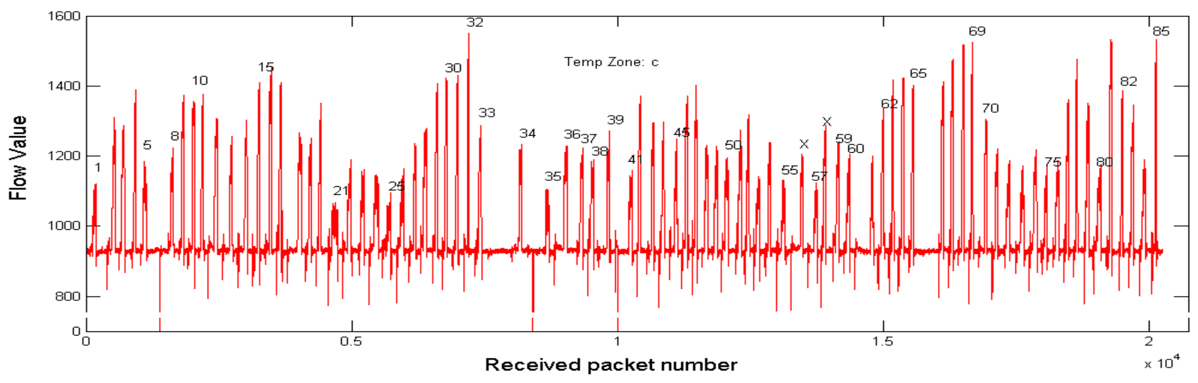


Figure 29: Flow Value vs. Received Packet Number for Temp Zone **c** (test7)

The above plot is for temperature zone **c**: $21 \leq Temperature < 25$. The number of total test runs for this region amounts to 80. This test yielded questionable results and did not fit in to the pattern of the rest of the three temperature zones. This is illustrated graphically in the next section.

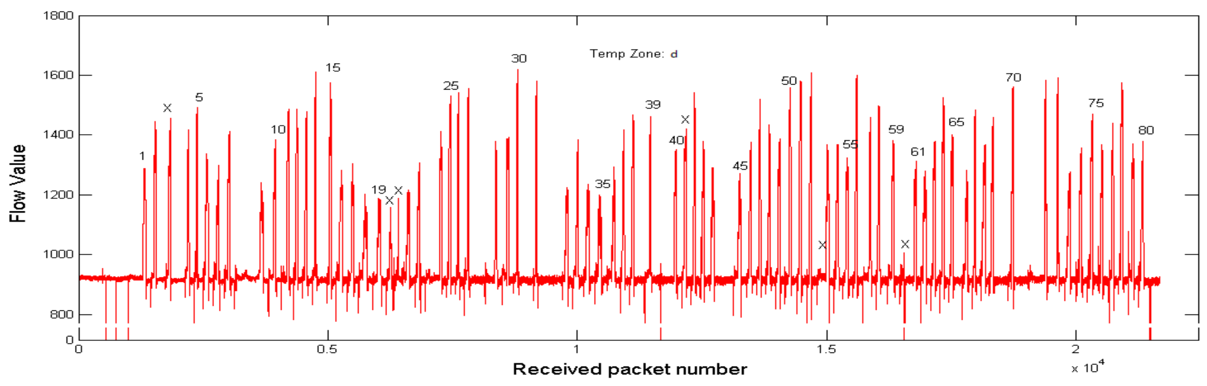


Figure 30: Flow Value vs. Received Packet Number for Temp Zone **d** (test10)

The above plot is for temperature zone **d**: $16 \leq Temperature < 21$. The number of total test runs for this region amounts to 74.

When the individual test run graphs are zoomed in (for example, Figure 4.3), many oscillations are visible. The reason for such behavior—mainly—has two reasons: non-maintenance of constant speed of the car and turbulence inside the channel. The latter is unavoidable unless a sophisticated channel is designed, but the maintenance of constant speed is very much possible with the proposed testbed in the earlier chapter.

5.2 Air Flow Speed vs. Flow Sensor Measurements Relation

The average air speed is calculated for each test run and mapped with its corresponding average flow value as described in Chapter 4.3. These values, plotted separately for each temperature zone, yields the following graph. The data points are shown with symbols * (in colors Magenta and Blue) and ∇. The curves show a non-linear trend. Therefore, a second order polynomial was selected for curve fitting, i.e. relation analysis (Equation 11).

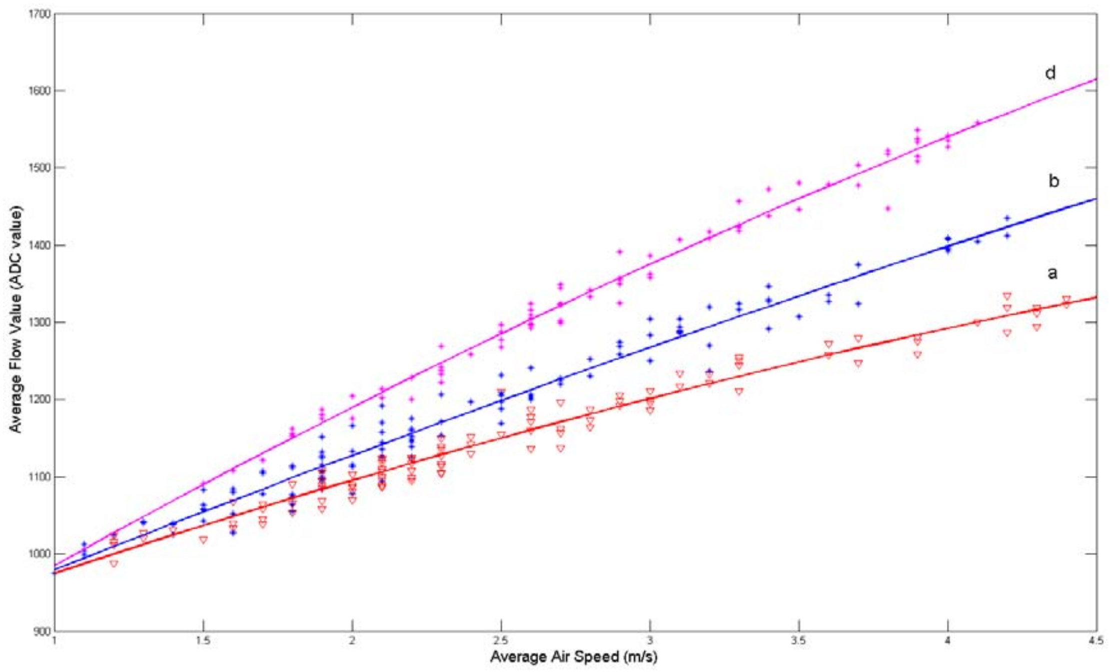


Figure 31: Average Air Speed (m/s) vs. Average Flow Value (ADC value)

Let the polynomial coefficients of temperature zones a, b, d be $[a]$, $[b]$, and $[d]$, respectively. The polynomial coefficients are given below.

a_0	839.1028
a_1	142.9233
a_2	-7.4201

b_0	822.1430
b_1	161.5691
b_2	-4.3923

d_0	759.6766
d_1	235.0738
d_2	-10.0030

Therefore, the relation of air speed and flow measurements can be written as follows for different temperature zones.

$$f^a : f^a(\nu) = -7.4201\nu^2 + 142.9233\nu + 839.1028 \tag{14}$$

$$f^b : f^b(\nu) = -4.3923\nu^2 + 161.5691\nu + 822.1430$$

$$f^d : f^d(\nu) = -10.0030\nu^2 + 235.0738\nu + 759.6766$$

where $f(\nu)$ represents the flow value and ν the air speed. When considering the offset values a_0, b_0, and d_0, the curves seem to intersect between $0 \leq \nu < 1$. It is false and such an implication can be due to non-precise coefficients, so it can be deduced that the curvature of the plots are greater than it should be. This is mainly due to the facts such as unavailability of flow samples for the region $0 \leq \nu < 1$ and inadequate number of samples for the rest of the region. Of course, turbulence in the channel plays a major role is adding to the errors of calibration. Such errors and human errors are difficult to quantify. However, the accumulated error—due to all unpredictable errors—is quantified and analyzed in the next section.

The relation in the temperature zone c also behaves like a second order polynomial. However, due to timing errors, it resulted in a non-compatible plot. It is shown below. Therefore, temperature zone c is not included in the above plot. It is compared with the temperature zones a, b, and d below.

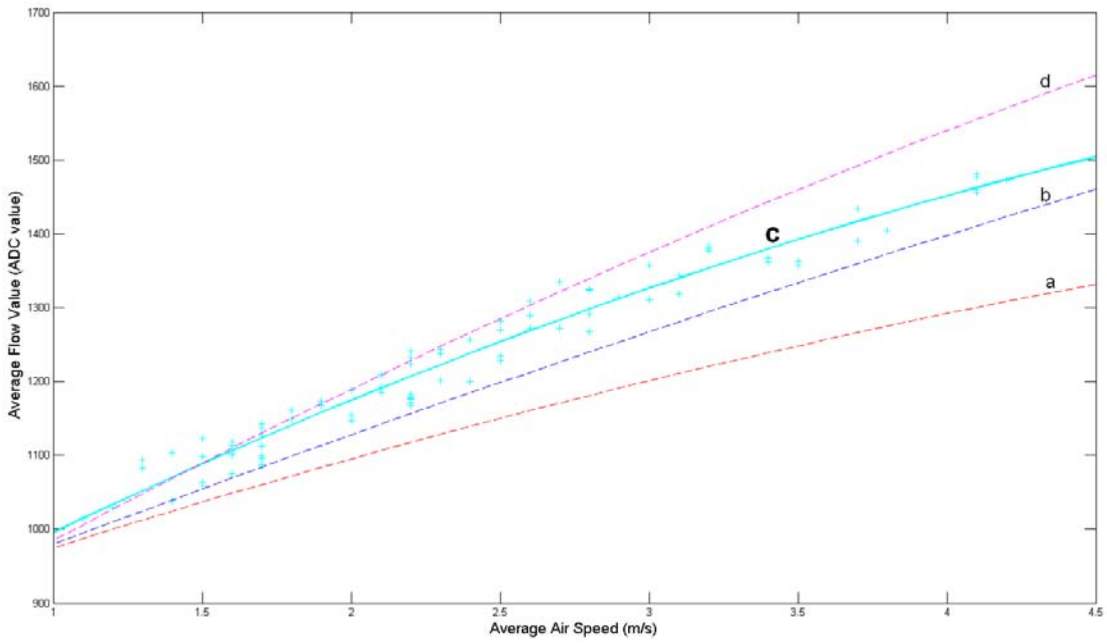


Figure 32: Temp Zone c plot (misfit)

5.2.1 Error Quantification

Error of curve fitting is calculated in the following manner:

$$e^j = f^j(\nu) - F^j \tag{15}$$

where j is the temperature zone, F experiment flow measurements, and f the flow values of the predicted curve. Plotting these three error vectors yields the following.

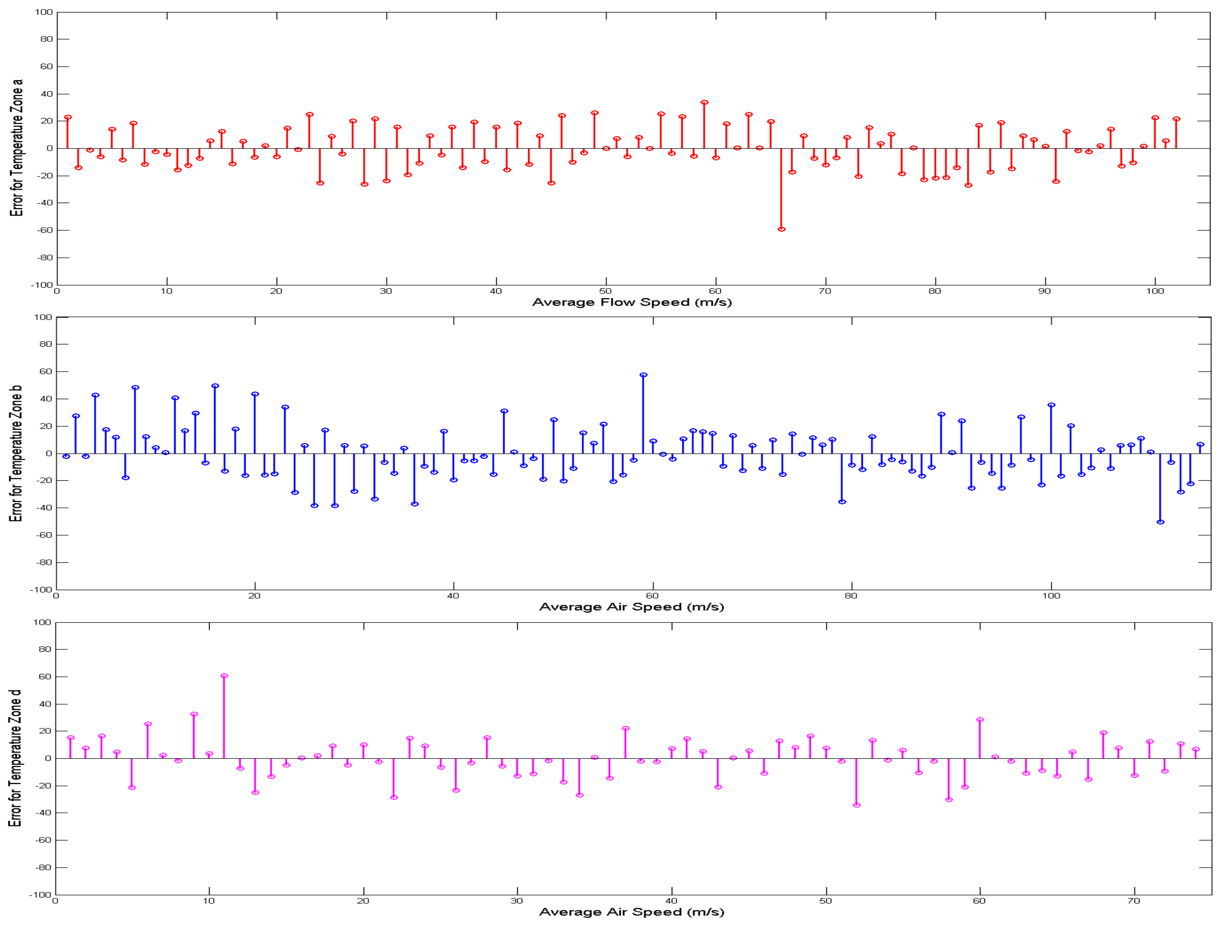


Figure 33: Curve Fitting Error

It is quite evident the oscillatory behavior of the error about zero-error. Only few high oscillations are found in all error plots. Therefore, given the drawbacks of the test method described earlier, the calculated air speed vs. flow relations equations prove to be adequate. A standard deviation can be calculated for these error vectors to assess the individual errors as a whole and to compare with all temperature zones.

$$Standard\,Deviation = \sqrt{\frac{sum\left(mean\left(\underline{e}^j\right) - \underline{e}^j\right)^2}{size\left(\underline{\nu}^j\right)}}$$

where j is the temperature zone, $\underline{e}$ the error vector, and $\underline{\nu}$ the average air speed vector for the given temperature zone. This produces standard deviations of 15.98, 21.52, and 13.38 for temperature zones a, b, and d, respectively. In comparison to each other, standard deviation's changes are on high side. This presents a bit of inconsistency between different temperature zones. However, it has to be noted that the temperature zone essentially means a collection of temperatures. Therefore, the heater temperature—though remaining within the zone—can vary up and down during the length of the experiment. Since flow measurements correlating to different air speeds are produced randomly, there is no direct way to indicate where such temperature variations may have occurred. But, it is collectively reflected in the standard deviation calculated above. Therefore, such temperature deviations within the zones can be the reason for showing unequal standard deviations.

However, the error is still too high for the flow sensor to be used for any precise work.

5.2.2 Inverted Flow Equation

The inverted flow equation calculates the speed of air flow based on the Equation 11 and the measured ADC reading of the flow (F). The Equation 11 can be rewritten for all temperature zones as a whole (Note: coefficients are changed to maintain generality):

$$f:\ F = p\nu^2 + q\nu + r$$

$$p\nu^2 + q\nu + (r - F) = 0$$

$$p\nu^2 + q\nu + \tilde{r} = 0$$

where $\tilde{r} = a_0 - F$. Since the equation is now reduced to a quadratic equation, the solutions are:

$$f^{-1}:\ \nu_{1,2} = \frac{-q \pm \sqrt{q^2 - 4p\tilde{r}}}{2p} \tag{16}$$

The above equation is applied for temperature zones a, b, and d, for the measured F values of the 3 temperature regions[13]. This above equation produces the estimated average speed of air. This value is compared with the actual average air speed values to produce the error, $Estimated\,Error = \nu_{estimate} - \nu_{actual}$. The standard deviation is calculated for this error estimation for all 3 temperature zones.

Temperature zone	Standard Deviation (m/s)
a	0.1559
b	0.1446
d	0.0914

Therefore, the final average air speed is as follows for different temperature zones:

$$\nu_a = \nu_{a,estimate} \pm 0.1559\ ms^{-1}$$

$$\nu_b = \nu_{b,estimate} \pm 0.1446\ ms^{-1}$$

$$\nu_d = \nu_{d,estimate} \pm 0.0914\ ms^{-1}$$

5.3 Additional Test for Temperature Zone e

The earlier flow measurements were taken at a constant temperature for all temperature zones (at 50^oC); therefore, ΔT varied for different temperature zones. Although this is a nonconventional approach, it yielded promising results. However, with this method, the calibration results still retains temperature dependence. This unnecessarily complicates the airflow speed calculation, as it requires different equations for different temperature regions.

[13]The Matlab code and error results can be found in the CD.

Another test was carried out with the same testbed for the temperature zone e, $11^oC \leqq T_e < 16^oC$. This test was done in the *conventional sense* of the constant temperature method: ΔT remains constant (25K) for all temperature zones under which the test is performed. With $\Delta T = 25K$, experimental results of temperature zone e is expected in and around temperature zone b, for tests in temp zone b was conducted inbetween $25^oC \leqq T_b < 30^oC$. This essentially means that $\Delta T \approx 25K$.

Temp zone e data:

Date	18.09.2010
Start Temperature	13.9^{o}C
End Temperature	13.3^{o}C
Test runs	98
Rejected test runs	9

The calculated relation between average flow and average speed for temp zone e is as follows (as per Chapter 5.2):

$$f^e :\ f^e(\nu) = -15.1579\nu^2 + 212.4405\nu + 778.0219$$

And, as per inverted equation principle in Chapter 5.2.2, the average air speed for temp zone e can be calculated as follows:

$$\nu_e = \nu_{e,estimate} \pm 0.2032\ ms^{-1}$$

The following graph shows the average air speed vs. average air flow relation, both the actual experimental values from test runs and the fitted curve (f^e).

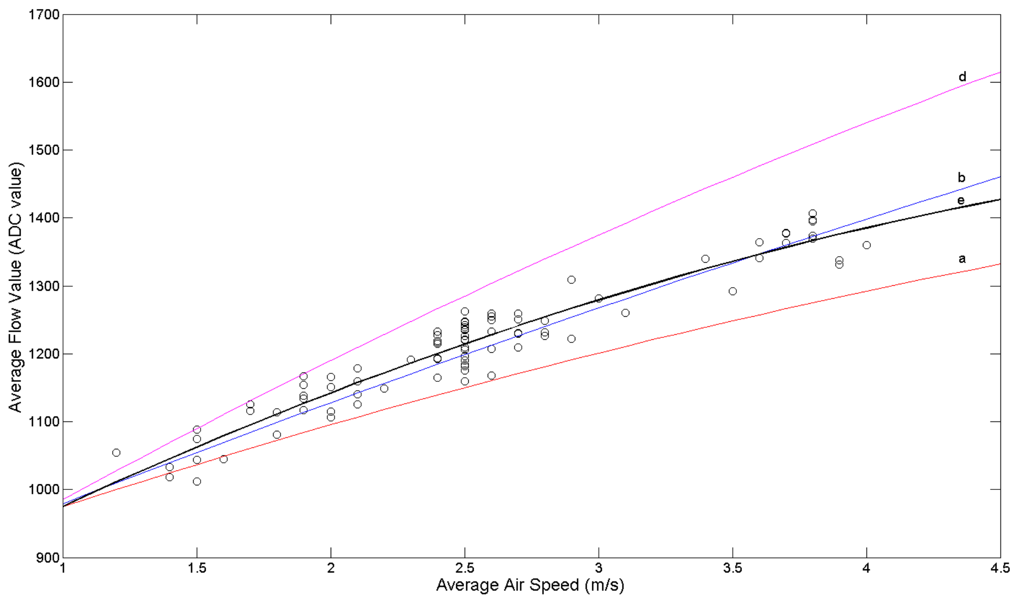


Figure 34: Average Air Speed vs. Average Flow for Temp Zone e

The fitted curve for temp zone e does not overlap the curve fitted for temp zone b. But, it is a near-fit, and the curvature of the fitted curve shows promise of overlapping with the curve for temp zone b.

The curvature of the e curve between air speeds 1.5 - 3.0 ms^{-1} is accentuated. Evidently, there are lot of measurements for that period, whereas flow measurements for air speeds less than 1.5 ms^{-1} and greater than 3 $ms^{-1}$are quite low. If more measurements are taken for the latter two regions of the graph, the curve will shape up from the second region side (speeds greater than 3 ms^{-1} – larger region than the region of speeds less than 1.5 ms^{-1}). Therefore, it is clear that the curves of e and b closely overlap. However, it will not be an exact match due to errors explained in previous sections; another major factor is that the temp zone b test consisted of 4 different tests conducted on 4 different days. There actual starting temperatures were 25.01, 26.57, 28.29, and 29.03^{o}C; for temp zone e it was only a single test that was performed at 13.9^{o}C. Therefore, it is not possible to see a near-exact match of curves e and b, anyway, because ΔT varies from approximately 25K to 20K for curve b.

The test was conducted in twilight in deteriorating light conditions. Therefore, the human errors—specifically the timing of the car—were error prone. Therefore, it is possible to see rather large oscillations of the measurement points about the fitted curve for temp zone e. This is also proven by the standard deviation value calculated for the inverted equation above, a value of 0.2032, the largest deviation of all 5 temperature zones.

However, evidently, curves for e and b have high resemblence. This is the expected result for other temperature regions, too, had they been conducted with the conventional constant temperature method. Therefore, the flow sensor needs to be calibrated with the proposed advance testbed under the conventional constant temperature method. It is expected to exclude the dependency of temperature from the avg flow vs. avg speed relation and yield a single equation to be used under varying ambient temperatures.

5.4 Energy Consumption

The energy consumption of the circuit of the testbed was evaluated for the test application. Energy consumption during the measurement window for the whole circuit, radio, MAX4462 amplifier, and MAX1683 voltage doubler were measured.

The healter voltage is shown below (left). Notice the constant nature of the graph during its last 10 - 20 ms. This is where the flow measurements are taken. However, as per the drawbacks explained earlier in the digital method of balancing the bridge, noticeable ripples are visible in the constant part of the graph (right).

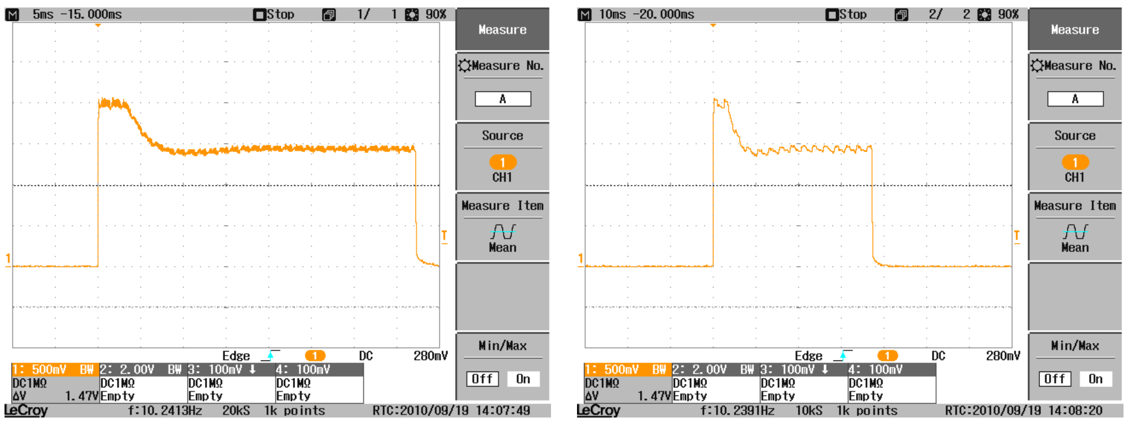
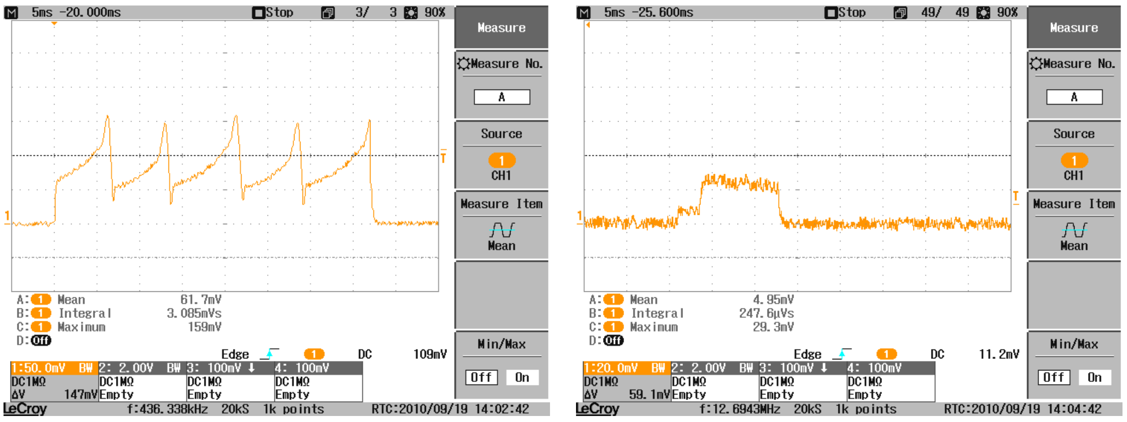


Figure 35: Heater Voltage (right: same graph showin ripple effect)

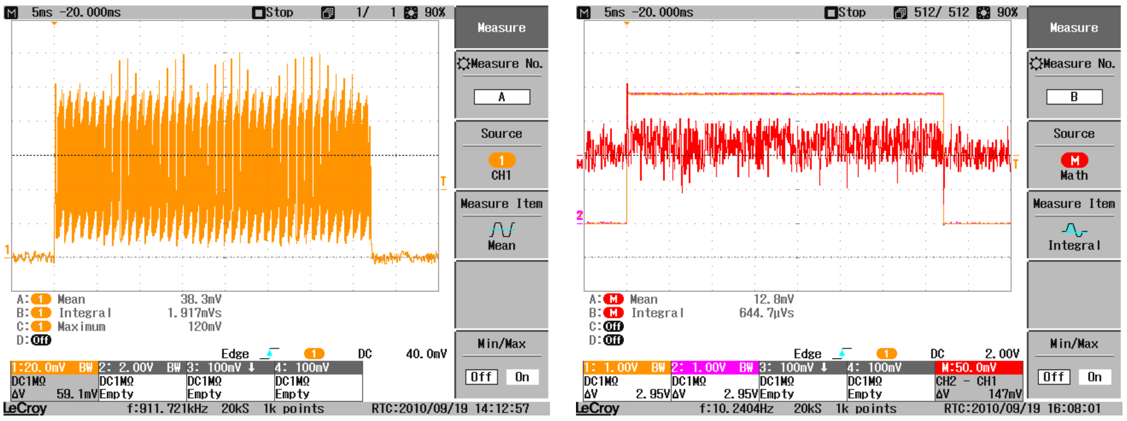
The voltage across the components was measured using a 1 Ω resistor. The MAX4462 needed to be measured using two probes (Note: one probe was a 10x probe while the other was 1x) to get the voltage differece across the resistor by calculating the difference of voltage in two probes. The used oscilloscope is capable of calculating the integral in mVs. However, since the voltage difference is divided by 1 Ω, it can be interpreted as mAs[14]. The values of mean current consumption is also give below for different components. However, note that the current consumption is averaged out for the measurement window shown in the graphs.



5ms -20.000ms Stop 3/ 3 90%
Measure
Measure No. A
Source CH1
Measure Item Mean
Min/Max Off On
A: Mean 61.7mV
B: Integral 3.085mVs
C: Maximum 159mV
D: Off
Edge DC 109mV
1:50.0mV BW 2: 2.00V BW 3: 100mV 4: 100mV
ΔV 147mV Empty Empty Empty
f:436.338kHz 20kS 1k points RTC:2010/09/19 14:02:42
5ms -25.600ms Stop 49/ 49 90%
Measure
Measure No. A
Source CH1
Measure Item Mean
Min/Max Off On
A: Mean 4.95mV
B: Integral 247.6µVs
C: Maximum 29.3mV
D: Off
Edge DC 11.2mV
1:20.0mV BW 2: 2.00V BW 3: 100mV 4: 100mV
ΔV 59.1mV Empty Empty Empty
f:12.6943MHz 20kS 1k points RTC:2010/09/19 14:04:42
LeCroy

Figure 36: Left: Total current; Right: Radio current

In each cylce of the test application, a single flow measurement is taken and then transmitted via the radio. The energy consumption of the radio is approx. 0.25 mVs (current approx. 20 mA). The total energy consumption for the circuit during flow measurement was approx. 3.1 mVs (averaged current approx. 82 mA).



5ms -20.000ms Stop 1/ 1 90%
Measure
Measure No. A
Source CH1
Measure Item Mean
Min/Max Off On
A: Mean 38.3mV
B: Integral 1.917mVs
C: Maximum 120mV
D: Off
Edge DC 40.0mV
1:20.0mV BW 2: 2.00V BW 3: 100mV 4: 100mV
ΔV 59.1mV Empty Empty Empty
f:911.721kHz 20kS 1k points RTC:2010/09/19 14:12:57
5ms -20.000ms Stop 512/ 512 90%
Measure
Measure No. B
Source Math
Measure Item Integral
Min/Max Off On
A: Mean 12.8mV
B: Integral 644.7µVs
C: Off
D: Off
Edge DC 2.00V
1: 1.00V BW 2: 1.00V BW 3: 100mV 4: 100mV M:50.0mV
ΔV 2.95V ΔV 2.95V Empty Empty CH2 - CH1 ΔV 147mV
f:10.2404Hz 20kS 1k points RTC:2010/09/19 16:08:01
LeCroy

Figure 37: Left: MAX1683; Right: MAX4462

The MAX1683 voltage doubler's energy consumption was approx 1.9 mAs (average current approx. 51 mA), whereas the MAX4462 amplifier showed an energy consumption of around 0.65 mAs (average current approx. 17 mA).

[14]Disregard visible *mean values* of the graphs as it takes in to account the entire visible area on the oscilloscope.

6 CONCLUSION

The thermal flow sensor of IMSAS is an ideal sensor for implementation in the Intelligent Container project. Some calibration has already been conducted for liquids; for air, the calibration results are confined to the laboratory, near room temperature. At present, however, it is without a calibration technique for its operating temperature spectrum. Developing of the circuits and programs for a manual calibration technique was started last year. The target of this project was to expand the scope of the last project to obtain a calibration method applicable to a wider temperature region—and to run the system autonomously on a wireless sensor mote.

The center point of the research was to automate the system. Automation depended mainly on the programming of the wireless mote on TinyOS operating system, using NesC programming language. This wireless communication specific programming required one-third of the work of the project. Another area, which required specific concentration, was the design and manufacture of a channel. A channel for the air to pass through had to be redesigned to maintain laminar flow. Several changes were made to the circuit to automate the manual balance technique used in the previous design. The bulk of the work was spent on testing of the autonomous calibration method using a novel concept, a toy car with speed control to simulate air speeds. Finally, a real-time application was developed to display the flow measurements. And, another set of offline applications were developed to analyze the test results.

Research course had to be adapted mid way to accommodate the compensation of new findings. However, at large, it remained unchanged. The problems encountered were numerous: timing issues with the digital potentiometer, autonomous balance technique, testbed and test methodology formation, and testing under different ambient temperatures. However, research managed to avert the presented complications. The calibration method detailed in this report contained some undesired errors, which are detailed in the results chapter. However, given the drawbacks of the testing methodology, they are within acceptable limits.

The used method can be used as a prototype in the Intelligent Container project. Beforehand, it is advisable to conduct more tests on the temperature regions that have already been covered and on regions that are not covered. This helps to improve the preciseness of the calibration curves.

The drawbacks of using a toy car were evident during testing. Therefore, a new test methodology is suggested in the Test Methodology section. The given system can be effectively used to gather a large amount of data—as oppose to the time consuming nature of the method used here—within a short period of time. This is a huge advantage as it reduces the time spent on test cycles. Some other changes are also proposed. Currently, the system runs on a 4 V battery. However, the motes used in the Intelligent Container project runs on 2 AA batteries (3 V). Circuit has to undergo few modifications to comply with this voltage. Further, more work is needed to develop an algorithm to network the spatial flow measurements taken in a container. One important parameter to be considered in this algorithm is the direction of flow in a three dimensional space. A strategic method to use as less flow sensors as possible to measure the three dimensional direction of an air flow in a container is the most desired.

The research presented here can be reused in future tasks and are so developed with well documented programs. The scope of the calibration in this project was limited to air flows at low speed. Given the proposed work is applied and the system redeveloped, it can be used to calibrate the sensor at high speeds, too. This essentially means that the sensor can be applied to other commercial tasks than just the research of the Intelligent Container project.

BIBLIOGRAPHY

[1] C. Lloyd. Integrating Thermal Flow Sensor to Telosb Module (mini project), University of Bremen, 2009.

[2] Microsystems Center Bremen. Intelligent Container. http://www.intelligentcontainer.com, 2010.

[3] A. Wessels. 0SGi Software-Bundles zur mobilen Überwachung von Waren mit Sensoren und RFID. Master's thesis, Bremen, 2009.

[4] R. Buchner, K. Rohloff, W. Benecke, and W. Lang. A high-temperature thermopile fabrication process for thermal flow sensors. volume 1, pages 575–578, June 2005. doi: 10.1109/SENSOR.2005.1496482.

[5] R. Buchner, M. Maiwald, C. Sosna, T. Schary, W. Benecke, and W. Lang. Miniaturised thermal flow sensors for rough environments. pages 582–585, 2006. doi: 10.1109/MEMSYS.2006.1627866.

[6] R. Buchner, C. Sosna, W. Benecke, and W. Lang. New integration technologies and applications for miniaturised thermoelectric flow sensors. volume 13, 22 - 24 May 2007.

[7] Analog Devices Inc. The ADG734BRUZ Datasheet. http://www.analog.com/static/imported-files/data_sheets/ADG733_734.pdf, 2010.

[8] Maxim Integrated Products Inc. The MAX5483 Datasheet. http://datasheets.maxim-ic.com/en/ds/MAX5481-MAX5484.pdf, 2010.

[9] Analog Devices Inc. The op90 datasheet. http://www.analog.com/static/imported-files/datasheets/OP90.pdf, 2009.

[10] The Mathworks. The Matlab Technical Document Help. http://www.mathworks.com/access/helpdesk/help/techdoc/ref/polyfit.html, 2010.

Appendix I

Mini project

The following figure shows the circuit used in the mini-project work conducted previously.

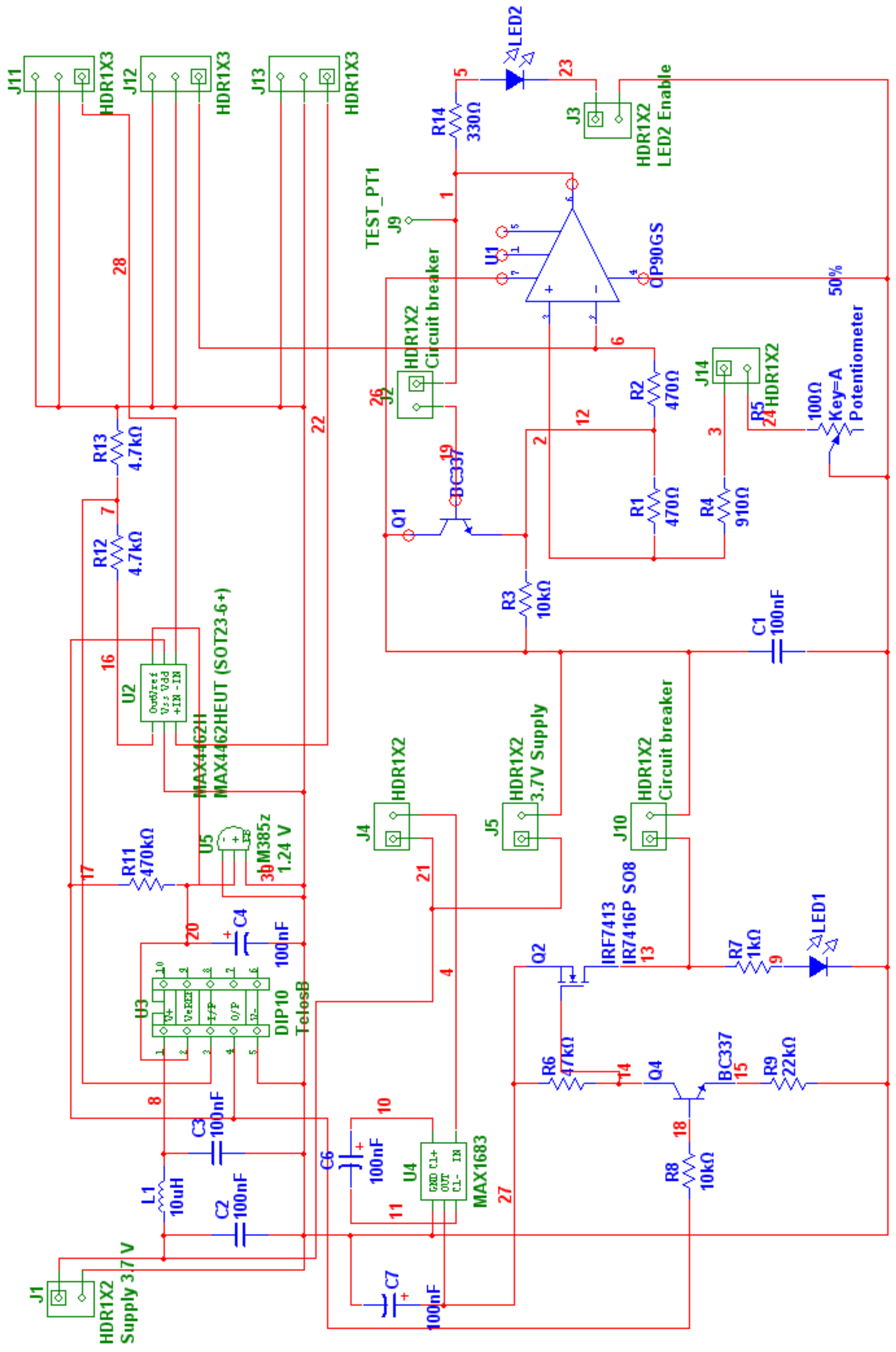


Figure 38: Mini-project circuit diagram

NesC

“nesC (network embedded systems C) is a component-based, event-driven programming language used to build applications for the TinyOS platform. TinyOS is an operating environment designed to run on embedded devices used in distributed Wireless Sensor Networks. nesC is built as an extension to the C programming language with components "wired" together to run applications on TinyOS."[15]

TinyOS

The following are extracts from Wikipedia

TinyOS is a free and open source component-based operating system and platform targeting wireless sensor networks (WSNs). TinyOS is an embedded operating system written in the nesC programming language as a set of cooperating tasks and processes. It is intended to be incorporated into smartdust. TinyOS started as a collaboration between the University of California, Berkeley in co-operation with Intel Research and Crossbow Technology, and has since grown to be an international consortium, the TinyOS Alliance.

TinyOS applications are written in NesC, a dialect of the C programming language optimized for the memory limitations of sensor networks. Its supplementary tools are mainly in the form of Java and shell script front-ends. Associated libraries and tools, such as the NesC compiler and Atmel AVR binutils toolchains, are mostly written in C.

TinyOS programs are built out of software components, some of which present hardware abstractions. Components are connected to each other using interfaces. TinyOS provides interfaces and components for common abstractions such as packet communication, routing, sensing, actuation and storage[16].

[15]Extract from http://en.wikipedia.org/wiki/NesC
[16]Extract from http://en.wikipedia.org/wiki/Tinyos

Appendix II

TelosB Program - NesC Files

The programming code (written in NesC) for the TelosB mote to read air flow values. The large number of files is essentially due to the inherent layered architecture of TinyOS and the inclusion of several abstractions over a driver. It is possible to reduce the number of files to achieve the same functionality. However, that substantially brings down the clarity and understandability.

File name	Description
ADG1_Pin_C.nc	Configures and wires pin 1 of U28 as a digital pin.
ADG1_Pin_P.nc	Configures pin 1 of U28 as an output pin.
ADG2_Pin_C.nc	Configures and wires pin 2 of U28 as a digital pin.
ADG2_Pin_P.nc	Configures pin 2 of U28 as an output pin.
BatteryAdc_C.nc	Wires BatteryAdc_P.nc to system ADC read client.
BatteryAdc_P.nc	Configures supply voltage half channel as an ADC channel to read battery voltage.
ControllerAppC.nc	Wires all user defined interfaces and modules to the system components.
ControllerC.nc	Program is booted in here. Governs the main timers for measurements and commands the program flow.
Digipot_CS_Pin_C.nc	Configures and wires pin 3 of U2 as a digital pin. This is the digital high/low required by digital potentiometer to latch the written data to its memory.
Digipot_CS_Pin_P.nc	Configures pin 3 of U2 as an output pin.
DigitalPotentiomater.nc	Interface command to write to the digital potentiometer.
DigitalPotentiomaterC.nc	Provides the write command of DigitalPotentiomater.nc interface.
FlowAdc_C.nc	Wires FlowAdc_P.nc to system ADC read client.
FlowAdc_P.nc	Configures pin 5 of U2 as an analog input pin.
FlowControl.nc	Interface command and events of flow sensor related functions.
FlowPin_C.nc	Configures and wires pin 7 of U2 as a digital pin. This is the Read Signal for flow measurement.
FlowPin_P.nc	Configures pin 7 of U2 as an output pin.
FlowSensorControl.nc	Provides all commands and events of interface FlowControl.nc and SensorControl.nc
Makefile	Provides the make command and pre-processor flags for compilation.
OP90_Adc_C.nc	Wires OP90_Adc_P.nc to system ADC read client. This is the pic to read the output of the voltage divider connected to OP90 output pin 6.
OP90_Adc_P.nc	Configures pin 10 of U2 as an analog input pin.
SensorControl.nc	Interface command and events of TelosB mote and radio related functionality.
ShtHumC.nc	Connects onboard Sensirion Sht11 temperature sensor.
ShtTempC.nc	Connects onboard Sensirion Sht11 humidity sensor.
SensingData.h	Contains all system related constants and radio packet structure.

Table 8: NesC programming files - General Application

The following mesh depicts the relations and wiring of files in Table 6.

Channel 2D

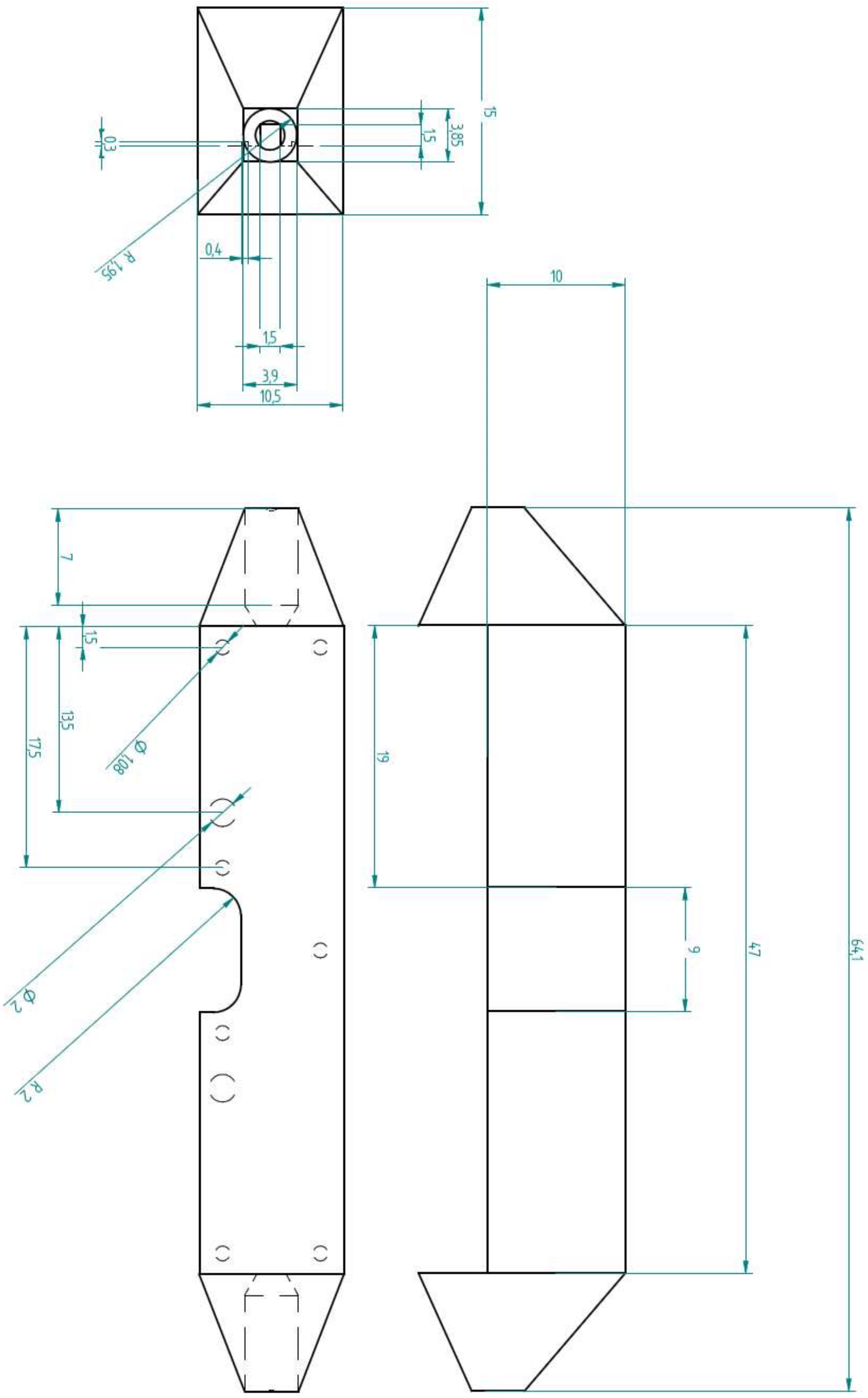


Figure 41: Channel 2D