Оглавление

1. Вводная часть

1.1. Введение

В современной промышленности все более строгие требования предъявляются к точности измерения параметров технологических процессов. При этом, для обеспечения точности исполнения тех или иных технологических операций необходим оперативный инструментальный контроль полуфабриката, для настройки технологического оборудования.

Кроме того, имеет место тенденция к использованию микропроб на этапе научно-исследовательской работы для контроля и управления потока- ми вещества в процессах, протекающих на микро- и наноуровневых структурах. К ведущим направлениям развития технологий относят: материаловедение, биотехнологии и инженерия. При реализации инструментальных средств, для таких технологий, необходимо вносить изменения в устройства контроля, так как погрешности, связанные с температурным дрейфом, оказывают существенное влияние на результат измерения.

Технологические процессы в различных областях производства связаны с подачей газа или его выделением. При этом контроль расхода газа позволяет судить о режиме протекания технологического процесса. Поэтому проблема повышения точности измерения расхода газа имеет первостепенное значение.

В зависимости от количества расхода газов применяют те или иные методы и средства измерений. Особую область в этих измерениях занимает контроль малых и сверхмалых объёмов, для которого непригодны механические, электрические, ультразвуковые и другие методы. Наибольшее распространение при контроле малых и сверхмалых объёмов газа получили пузырьковые методы. Существующие пузырьковые методы отличаются учётом различных факторов, влияющих на содержание газа в каждом пузырьке, так например, при реализации метода Т.С. Бондарева и В.С. Малышева осуществляется барботаж измеряемого газа через слой жидкости с одновременным подсчетом числа пузырьков газа. Однако, в подобных дискретных устройствах отсутствует учет параметров жидкой среды, используемой для барботирования. Сложность заключатся во влиянии, изменяющейся температуры барботажной жидкости, на величину коэффициента поверхностного натяжения и на объём газа в пузырьке при его образовании у сопла на границе жидкой и газообразной фаз.

При этом, не учитывается давление газа в пузырьке, и поэтому в определение расхода газа вносится значительная погрешность. При более точных измерениях необходимо учитывать реологические свойства жидкости через которую происходит барботирование газа, температуру и т.д. Это накладывает ограничения при проведении измерения, связанные с необходимостью термостатирования, и снижает возможности метода и средства измерения.

Процессы, связанные с выделением малых объёмов газа, широко распространены. Например, в области пищевой промышленности от качества дрожжей зависит качество хлебопекарной продукции. В компрессорной технике такие процессы применимы к определению расхода газа в микрокомпрессорах.

Объект исследования: поток газовых пузырьков, движущийся в средствах измерения расхода газа пузырьковым методом.

Предмет исследования: дискретный пузырьковый метод и средство измерения малых расходов газа, учитывающие реологические свойства среды для барботирования.

Целью бакалаврской работы является повышение точности измерения расхода сверхмалых объёмов газа, путём разработки метода и технических средств, обеспечивающих контроль указанного расхода с учётом давления жидкости и её реологических свойств.

Методы и средства исследования. При решении поставленных задач применялись теория физико-химической механики и реологии сред, математический аппарат теории вероятностей и математической статистики, интегрального и дифференциального исчисления, теории численного анализа, методы математического моделирования на ЭВМ, реализованные в оболочках Lab VIEW,MathCAD.

Достоверность полученных результатов. Достоверность результатов работы подтверждается корректным использованием теоретических и экспериментальных методов обоснования полученных результатов и выводов. Достоверность экспериментальных данных обеспечивается использованием современных средств и методик проведения исследований. Положения теории основываются на известных достижениях в области молекулярной физики (1), гидродинамики ламинарного течения жидкости и движения тел в жидкостях.

Положения, выносимые на защиту:

Метод измерения сверхмалых расходов газа, определение конструктивных особенностей измерительной кюветы.

Математическая модель процесса барботирования газа через жидкость, отличающаяся учётом реологических параметров вязкой среды.

Алгоритмы распознавания образов, использующиеся при распознавании пузырьков газа, проходящих через жидкостную среду.

1.2. Обзор методов и средств измерений малых расходов газа

Источник (2), регламентирует разновидности расходомеров и их преобразователей.

В настоящее время потребности в приборах для измерения расхода и объема различных продуктов удовлетворяются в основном общепромышленными приборами и устройствами. Приборы для измерения расхода, массы или объема продуктов должны обладать высокой точностью и надежностью измерения, так как большинство измерений являются учетно-отчетными и на основании их производятся приемка и сдача исходного сырья или готового продукта.

Нормальная эксплуатация всех типов приборов возможна лишь при соблюдении правил эксплуатации, основными из которых являются: поддержание температуры и давления измеряемой среды в допустимых пределах; соответствие плотности и вязкости измеряемой среды градуировочным.

Измерение малых расходов жидкостей и газов необходимо учитывать при создании различных полузаводских установок, при проведении многих. видов научно-исследовательских работ и при контроле промышленных процессов.

Принято верхнюю границу малых расходов Q_{MAX} связывать с соответствующей верхней границей диаметра трубопровода $D_в$. Если принять $D_в$ = 10 мм, то Q_{MAX} для жидкостей будет 1 м3/ч, а для газов - 10 м3/ч. Если же за верхнюю границу принять Dв = 5 мм, то для жидкостей - Q_{MAX} 0,25 м3/ч, для газов - Q_{MAX}, 2,5 м3/ч.

Нижняя же граница малых расходов определяется требованиями практики: например, 1 $см^3$/ч для жидкостей и 50 $см^3$/ч для газов. Для измерения малых расходов применяются особые разновидности методов измерения и, кроме того, некоторые специальные методы.

Расход вещества - это масса или объем вещества, проходящего через данное сечение канала средства измерения расхода в единицу времени. В зависимости от того, в каких единицах измеряется расход, различают объемный расход или массовый расход.

Объемный расход измеряется в м3/с (м3/ч, л/с и т.д.), а массовый - в кг/с (кг/ч, т/ч и т.д.).

Расход вещества измеряется с помощью расходомеров, представляющих собой средства измерений или измерительные приборы расхода. Для измерения малых расходов газа используются следующие основные группы расходомеров: переменного перепада давления; обтекания постоянного перепада давления; тахометрические; электромагнитные; переменного уровня; тепловые; вихревые;

акустические; кориолисовые силовые; капельно-пузырьковые. Все существующее многообразие методов и средств, можно разделить на две большие группы, рисунок 1.1.

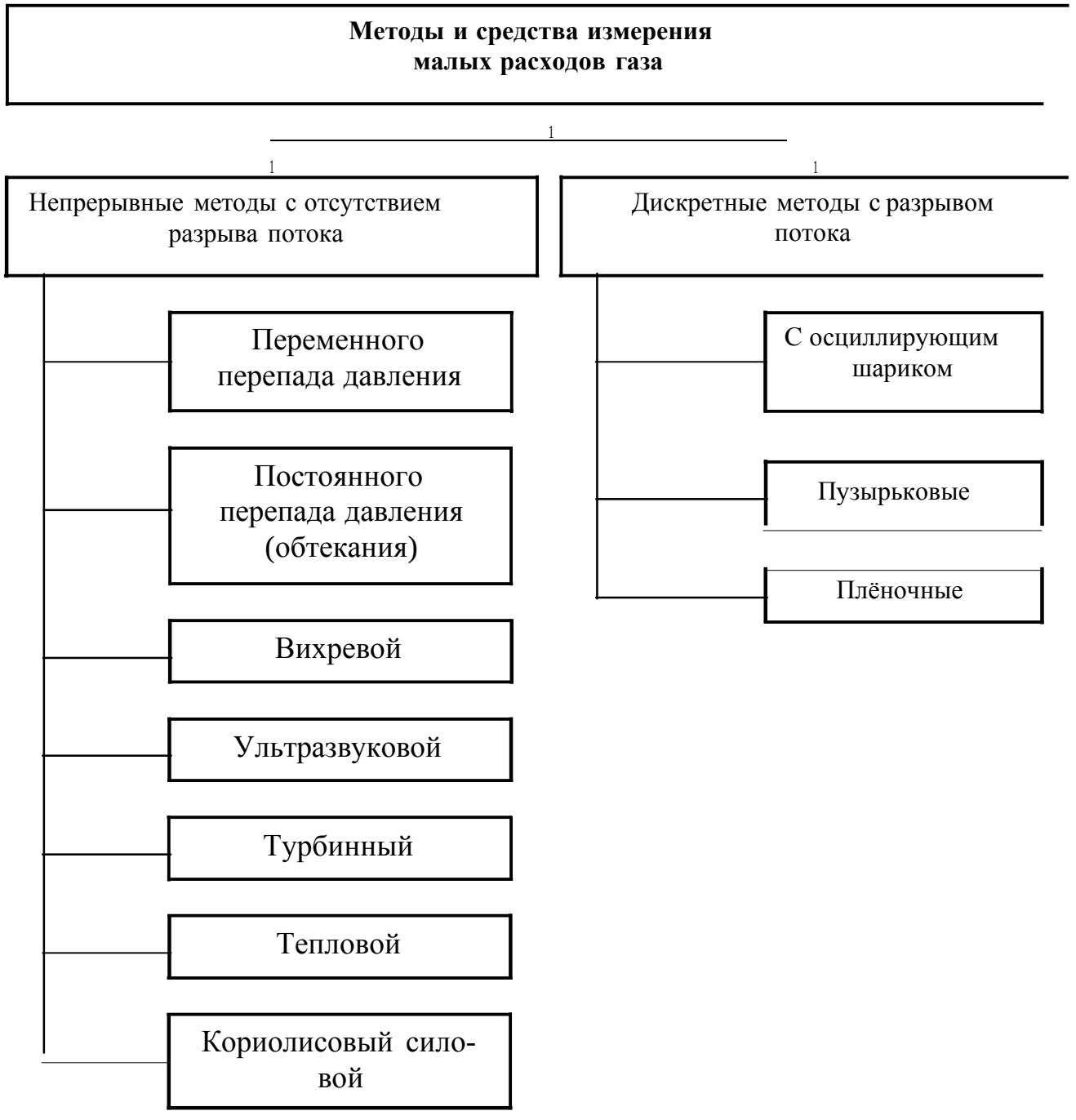


Рисунок 1.1 - Расходомеры применяемые для измерения малых расходов газа

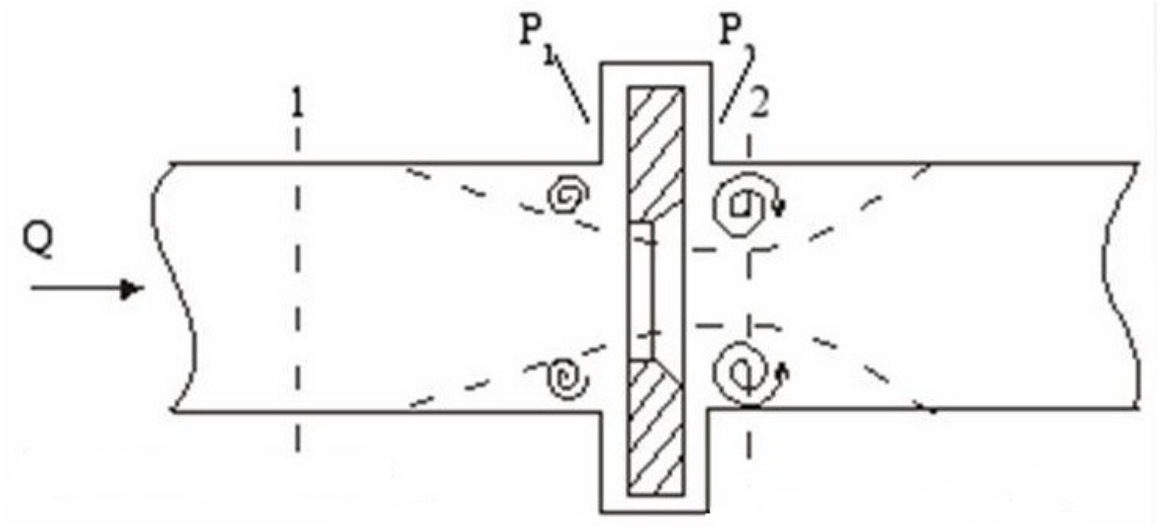


Рисунок 1.2. – Расходомер переменного перепада давления

В соответствии с рисунком 1.2, измерительная диафрагма 2 является первичным измерительным преобразователем расхода и представляет собой диск, установленный так, что центр его лежит на оси трубопровода. При протекании потока в трубопроводе с диафрагмой он сужается. Перед диафрагмой и после нее образуются зоны завихрения. Давление струи около стенки вначале возрастает из-за подпора перед диафрагмой. За диафрагмой оно снижается до минимума, затем снова повышается, но не достигает прежнего значения, так как вследствие трения и завихрений происходит потеря давления $P_{пот}$. Таким образом, часть потенциальной энергии давления потока переходит в кинетическую. В результате средняя скорость потока в суженном сечении повышается, а статическое давление в этом сечении становится меньше статического давления перед сужающим устройством. Разность этих давлений (перепад давления) служит мерой расхода протекающей через сужающее устройство жидкости, газа или пара.

В качестве измерительных приборов применяются различные дифференциальные манометры 3, снабженные мембраной 5, индуктивным преобразователем 4, 6 обеспечивающим выдачу измерительной информации о расходе в виде переменного электрического сигнала.

В расходомерах обтекания (постоянного перепада давления, рисунок 1.3) принцип действия основан на том, что обтекаемое тело (поплавок, поршень, шарик) воспринимает динамическое давление со стороны обтекающего его потока. При этом возрастание расхода увеличивает вертикальное перемещения тела-поплавка, находящегося в потоке и изменяющего при этом площадь проходного отверстия прибора таким образом, что перепад давления по обе стороны поплавка остается постоянным.

Расходомеры постоянного перепада давления - ротаметры - применяются для измерения расходов однородных потоков чистых и слабозагрязненных жидкостей и газов, протекающих по трубопроводам и не подверженных значительным колебаниям.

Ротаметр, изображённый на рисунке 1.3, представляет собой длинную коническую трубку 1, располагаемую вертикально, вдоль которой под действием движущегося снизу вверх потока перемещается поплавок 2.

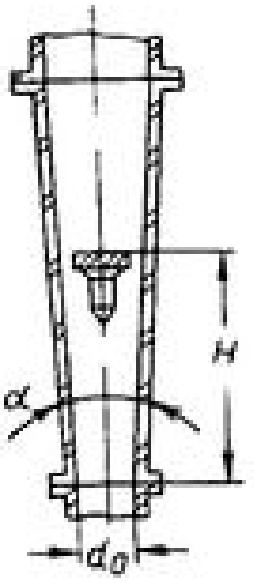


Рисунок 1.3 – Расходомер постоянного перепада давления - ротаметр

Поплавок перемещается до тех пор, пока площадь кольцевого отверстия между поплавком и внутренней поверхностью конусной трубки не достигнет такого размера, при котором перепад давления по обе стороны поплавка не станет равным расчетному. При этом действующие на поплавок силы уравновешиваются, а поплавок устанавливается на высоте, соответствующей определенному значению расхода.

В соответствии с рисунком 1.4, шарик 2, выполненный из магнитной коррозионно-стойкой стали, находящийся в стеклянной трубке 3, совершает вынужденные колебания вверх и вниз под воздействием электромагнитов 4, расположенных в верхней части трубки. Поднимаясь вверх, шарик в средней части трубки прерывает луч света, падающий от осветителя 1 на фотоэлемент 8. Это приводит через генератор импульсов 7 и усилительное устройство 5 к выключению электромагнитов, и шарик 2 падает на упор. Затем цикл повторяется. Чем больше расход жидкости, поступающей в трубку снизу, тем медленнее происходит падение шара и тем меньше частота его колебаний, измеряемая прибором 6. Диапазон измерений серийно выпускаемого прибора на данном принципе составляет 0,05 – 1000 $см^3$/мин и обеспечивается набором сменных шариков и стеклянных трубок.

В настоящее время разработаны и имеют весьма широкие перспективы применения вихревые расходомеры («эффект фон Кармана»), принцип действия которых основан на

зависимости от расхода частоты колебаний давления среды, возникающих в потоке в процессе вихреобразования. (3)

Измерительный преобразователь вихревого расходомера (рисунок 1.5) представляет собой завихритель 1, вмонтированный в трубопровод, с помощьюкоторого поток завихряется (закручивается) и поступает в патрубок 2. На выходе из патрубка в расширяющейся области 4 установлен электроакустический преобразователь 3, воспринимающий и преобразующий вихревые колебания потока в электрический сигнал, который далее приводится к нормализованному виду.

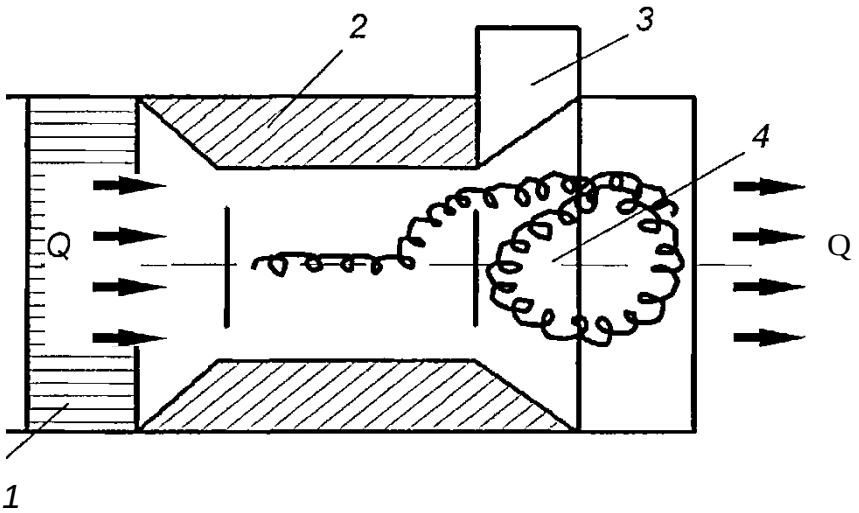


Рисунок 1.5 - Функциональная схема измерительного преобразователя вихревого расходомера

Завихрения потока формируются таким образом, что внутренняя область вихря - ядро, поступая в патрубок 2, совершает только вращательное движение. На выходе же из патрубка в расширяющуюся область 4 ядро теряет устойчивость и начинает асимметрично вращаться вокруг оси патрубка.

В некоторых случаях для удаления вихрей и исправления движения потока требуются выпрямляющие клапаны или прохождение потока по прямой трубе определенной длинны. В потоке со слабым давлением вихри образуются нерегулярно, и такие условия создают трудности при использовании вихревых расходомеров. Точность вихревых расходомеров достаточна для практических (технических) измерений.

Идеальными условиями для применения указанных расходомеров являются потоки со средней или высокой скоростью, поскольку при малых объёмных расходах вихри образуются нерегулярно. Поэтому их применяют преимущественно для измерения промышленных расходов газов (при скоростях 10 - 15 м/с и диаметрах условного прохода преобразователя 25 - 50 мм).

К преимуществам электроакустических расходомеров относится бесконтактность измерений, отсутствие движущихся частей в потоке, отсутствие потерь давления в трубопроводах и др.

Принцип действия акустических расходомеров основан на зависимости акустического эффекта в потоке от расхода веществ. Известно несколько методов использования звуковых (как правило, ультразвуковых) колебаний для измерения расходов жидкостей и газов. Один из них, так называемый «фазовый», основан на том, что при распространении звуковой волны в движущейся среде время ее прохождения от источника до приемника определяется не только скоростью распространения звука в данной среде, но и скоростью движения самой среды.

Акустический расходомер, работающий по двухканальной фазовой схеме (рисунок 1.6), состоит из ультразвукового генератора 1, питающего излучающие пьезопреобразователи 2 и 3; приемных пьезопреобразователей 4 и 5; фазовращающего устройства 6 для устранения путем асимметрии каналов преобразователей возникающих фазовых сдвигов; электронного усилителя 7 и измерительного прибора 8, который градуируется в единицах расхода.

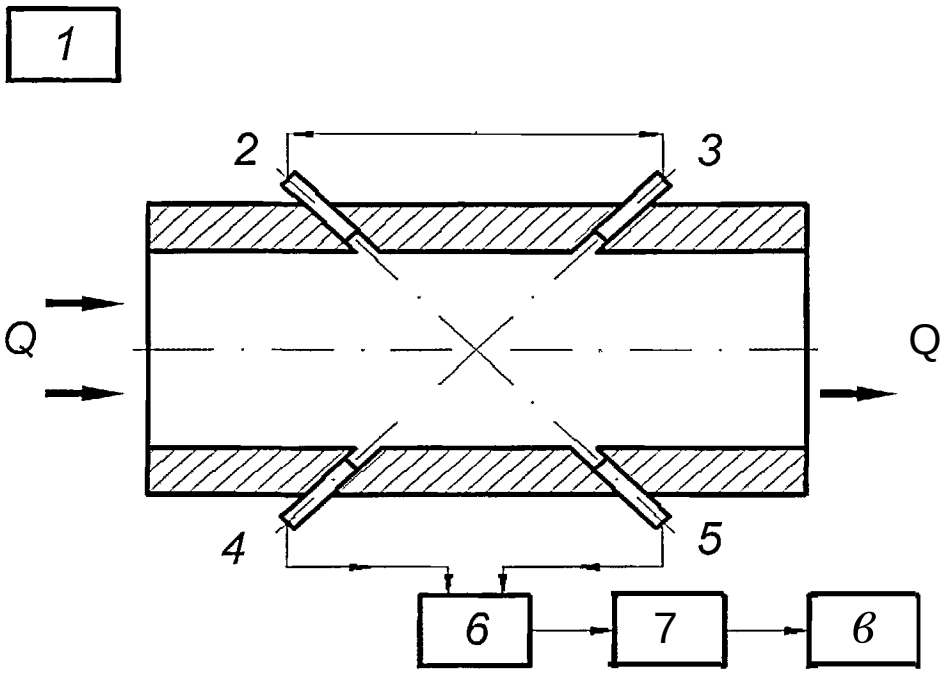


Рисунок 1.6-Структурная схема акустического (ультразвукового) расходомера

Если звуковая волна направлена по движению потока, скорости их складываются, если против потока, - вычитаются. Разность времени прохождения звука по направлению потока и против него пропорциональна скорости потока, а следовательно, расходу протекающей жидкости.

В последнее время получают распространение ультразвуковые расходомеры, в которых используется эффект Допплера, заключающийся в том, что ультразвуковые волны,

генерируемые излучателями, отражаются от взвешенных частиц, завихрений, пузырьков газа и т. п. в потоке измеряемой среды и воспринимаются приемниками отраженных излучений. Разность между частотами излучаемых и отраженных акустических волн позволяет определить скорость потока. Временные ультразвуковые расходомеры применяются в чистых газах без завихрений.

В турбинных расходомерах чувствительным элементом является аксиальная или тангенциальная турбина, приводимая во вращение потоком измеряемой среды. Принцип действия основан на том, что число оборотов турбины в единицу времени пропорционально скорости потока. Расходомеры турбинного типа, рисунок 1.7, работают с наименьшей погрешностью с постоянными потоками, и редко применяются при измерении малых расходов газов.

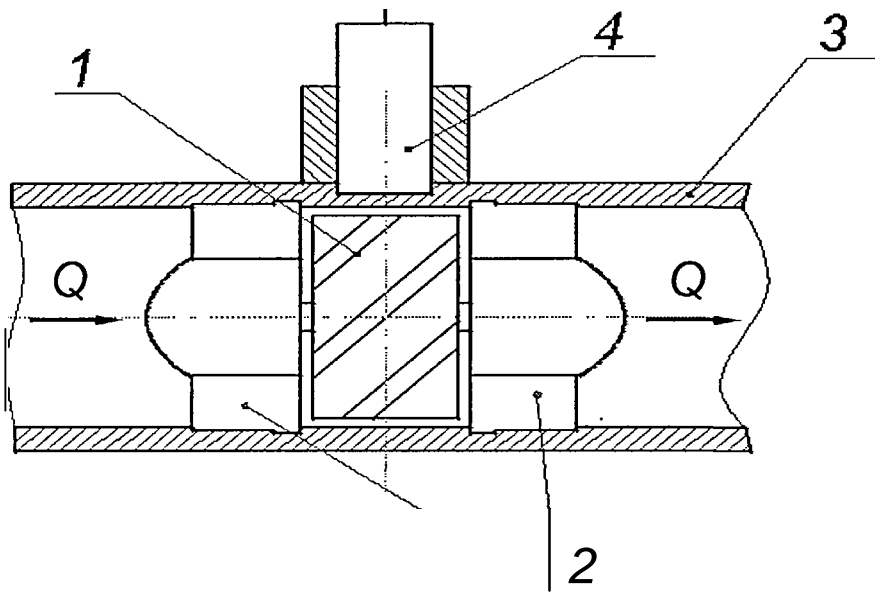


Рисунок 1.7 – Расходомер турбинный

В соответствии с рисунком 1.7, поток измеряемой среды приводит во вращение турбинку 1, установленную с помощью опор 2 в проставке 3. Число оборотов турбинки преобразуется датчиком 4 в импульсы напряжения. Импульсы от датчика оборотов поступают во вторичный прибор 5.

Вторичный прибор предназначен для преобразования импульсов датчика оборотов в единицы объёма и индикации разового, суммарного объёма и мгновенного расхода жидкости.

При точных измерениях важно, чтобы не происходило завихрения протекающего вещества, поскольку это напрямую сказывается на частоте вращения турбинки. Поэтому спрямляющие поток лопатки устанавливаются обычно на входе расходомера. Эти лопатки

формируют также одну из опорных точек турбинки. На рисунке 6 данными лопатками являются опоры. Высота турбины, как правило, не превышает 25-30% радиуса.

Одним из преимуществ турбинных расходомеров по сравнению с расходомерами других типов является линейная зависимость их выходного сигнала от скорости потока в установленном для прибора диапазоне.

Преобразование скорости вращения турбинки в объемные значения количества прошедшего газа осуществляется путем передачи вращения турбинки через магнитную муфту на счетный механизм, в котором путем подбора пар шестеренок (во время градуировки) обеспечивается линейная связь между скоростью вращением турбинки и количеством пройденного газа.

Другим методом получения результата количества пройденного газа в зависимости от скорости вращения турбинки является использование для индикации скорости магнитоиндукционного преобразователя. Лопатки турбинки при прохождении вблизи преобразователя возбуждают в нем электрический сигнал, поэтому скорость вращения турбинки и частота сигнала, поступающего с преобразователя, пропорциональны.

Тепловые расходомеры могут применяться при измерении небольших расходов практически любых сред при различных их параметрах. Тепловые расходомеры основаны на измерении зависящего от расхода эффекта теплового воздействия (нагрева или охлаждения) на поток или тело, контактирующее с потоком (4).

Тепловые расходомеры могут выполняться по трем основным принципиальным схемам:

калориметрические, основанные на нагреве или охлаждении потока посторонним источником энергии, создающим в потоке разность температур;

теплового слоя, основанные на вычислении разности температур с двух сторон пограничного слоя или мощности нагрева;

термоанемометрические, в которых используется зависимость между количеством теплоты, теряемой непрерывно нагреваемым телом, помещенным в поток, и массовым расходом вещества.

Выбор принципиальной схемы измерения зависит от измеряемой среды, необходимой точности, типа используемых термочувствительных элементов и режима нагрева. Для упруго-вязких пластичных веществ, какими являются опара и тесто, а также многие другие пищевые продукты, предпочтительным является измерение по схеме термоанемометра с постоянной температурой подогрева потока.

Пример, расходомера калориметрического типа приведён на рисунке 1.8.

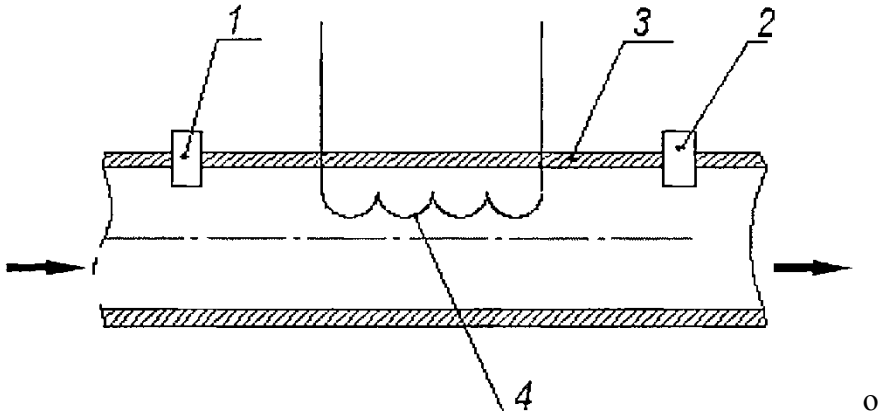
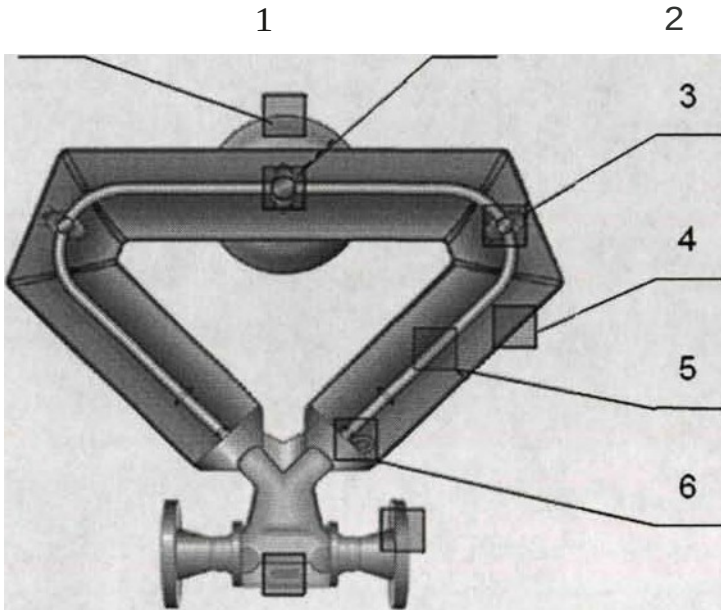


Рисунок 1.8 -Расходомер калориметрический

Расходомер приведённый на рисунке 1.8 состоит из нагревателя 4, расположенного внутри трубопровода 3 , и двух термопреобразователей 1 и 2 для измерения температур до и после нагревателя. Термопреобразователи располагаются обычно на равных расстояниях от нагревателя. Распределение температур по обе стороны от источника нагрева будет зависеть от расхода вещества. С помощью тепловых расходомеров может быть обеспечена точность измерения расхода вязких продуктов ±2 - 2,5%.

Сейчас всё более широкое распространение получают расходомеры с наружным расположением нагревателя и термопреобразователей. Указанные тепловые неконтактные расходомеры применяют также, при измерении малых расходов, как жидкостей, так и для газов. Обладая повышенной чувствительностью, именно в области малых и микрорасходов газов, приборы указанного типа являются конкурентно способными с современными расходомерами (например, кориолисовыми силовыми расходомерами, ультразвуковыми расходомерами).

Кориолисовые силовые расходомеры (5) состоят из одной или нескольких изогнутых вибрирующих трубок, рисунок 1.9. Газ, расход которого следует измерить, проходит по вибрирующим трубкам. Скорость движения газа увеличивается, когда она приближается к точке максимальной вибрации, и снижается, когда газ проходит эту точку. При этом трубке передается вращательное движение. Сила вращательного движения прямо пропорциональна массе расхода. На входе датчика расходомера газ разгоняется под воздействием собственной инерции, а затем снижает скорость и по мере прохождения через расходомер. Инерция газа создает силу Кориолиса, которая искривляет измерительную трубку. Степень искривления пропорциональна расходу.



1 – процессор, 2 – катушка возбуждения 3 - измерительная катушка; 4 – корпус; 5 – расходомерная трубка; 6 – терморезистор;

Рисунок 1.9 - Кориолисовый силовой расходомер

В соответствии с рисунком 1.9, катушка возбуждения 2 используется для генерирования колебаний расходомерных трубок 5, для поддержания вибрации трубок на их собственной частоте. Катушки измерительных преобразователей 3 представляют собой электромагнитные детекторы, расположенные с каждой стороны расходомерной трубки. Измерительные катушки 3 вырабатывают сигнал, отражающий скорость и положение в данной точке вибрирующей трубки, и масса потока определяется путем измерения разницы фаз между этими сигналами. Процессор 1 осуществляет измерение и обработку первичных сигналов, поступающих от измерительных катушек 3 и терморезистора 6. Температура трубки постоянно измеряется, поскольку её колебательные свойства изменяются в зависимости от температурных изменений. Благодаря использованию терморезистора 6 в измерения удается внести требуемые поправки.

Кориолисовые силовые расходомеры непосредственно измеряют массовый расход в отличие от других расходомеров, которые рассчитывают массовый расход, используя предполагаемое значение плотности. Умножив площадь сечения трубки на среднюю скорость газа, получаем объемный расход. Значение массового расхода определяется при умножении объемного расхода на плотность газа. Некоторые расходомеры, указанного типа, измеряют температуру и давление газа, и используют полученные значения для нахождения величины плотности газа.

Несмотря на высокую точность измерения, эти расходомеры имеют ограничения по диаметру труб, на которых они могут эффективно использоваться. Более 90% расходомеров Кориолиса применяются на трубах с диаметром менее 5 сантиметров. Кориолисовые силовые расходомеры применяются для измерения чистых газов, которые перемещаются со скоростью, достаточной для её измерения в трубах до 5 см.

В таблице 1.2 представлены сравнительные метрологические характеристики методов измерения расходов газов и отдельных типов расходомеров, а также ограничения на применение этих методов измерения.

Название метода измерения	Наименьший расход	Погрешность,	Марка прибора
Переменного перепада давления	10-200 мл/с	±1,5	СГКИ-4
Постоянного перепада давления (обтекания)	10-50 мл/с	±1-3	DK-32, РМ-1
Вихревой	1-40 м3/ч	±1	Ирга-РВ
Ультразвуково	20-50 мл/с	±2	ГАЗ 001
Турбинный	0,91-6,6 MJ/ч	±0,5	CRA
Тепловой	0,01- 10 мл/с 1-10 мл/с	±2,5 ±1	ИРГ, FlexMASSter
Кориолисовый силовой	0,2-5 г/ч 3-70 мл/мин	±0,4 ±0,7	miniCORI-FLOW, шini CORI-FLOW

Таблица 1.2 - Технические характеристики средств измерения расхода газа

Дискретные методы измерения расхода газа с разрывом потока

Расходомеры, основанные на процессах в двухфазных системах, связанные с образованием капли в газовой среде или пузырька в жидкости получили широкое распространение.

В основе функционирования пленочно-пузырьковых устройств, лежит свойство поверхностного натяжения. Известен индикатор малых расходов газа. Из рисунка 1.9, видно, что устройство состоит из капиллярной трубки 1, выходной конец которой заканчивается раструбом 2, а в капиллярный канал введена капля 3 смачивающей жидкости. Свойство поверхностного натяжения жидкости и простота такой конструкции позволяют обеспечить возможность непрерывной индикации расхода в условиях невесомости.

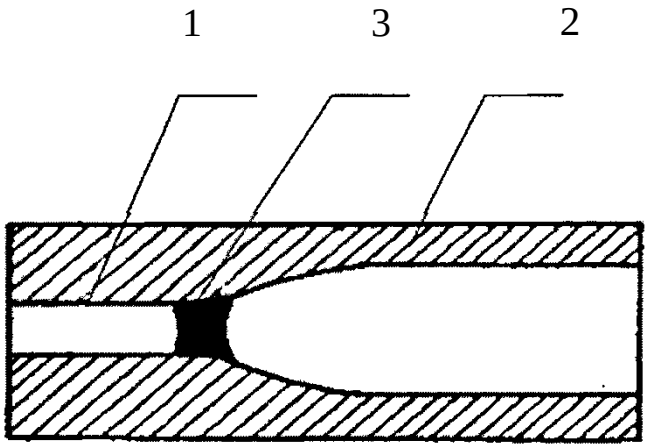


Рисунок 1.9 - Индикатор малых расходов газа

Расходомеры, в основе которых лежит процесс образования тонкой пленки, называются пленочными. Расходомеры такого типа получили свое распространение при измерении малых расходов газа в технологических процессах микроэлектронной промышленности при калибровке и поверке тепловых расходомеров, рисунок 1.10.

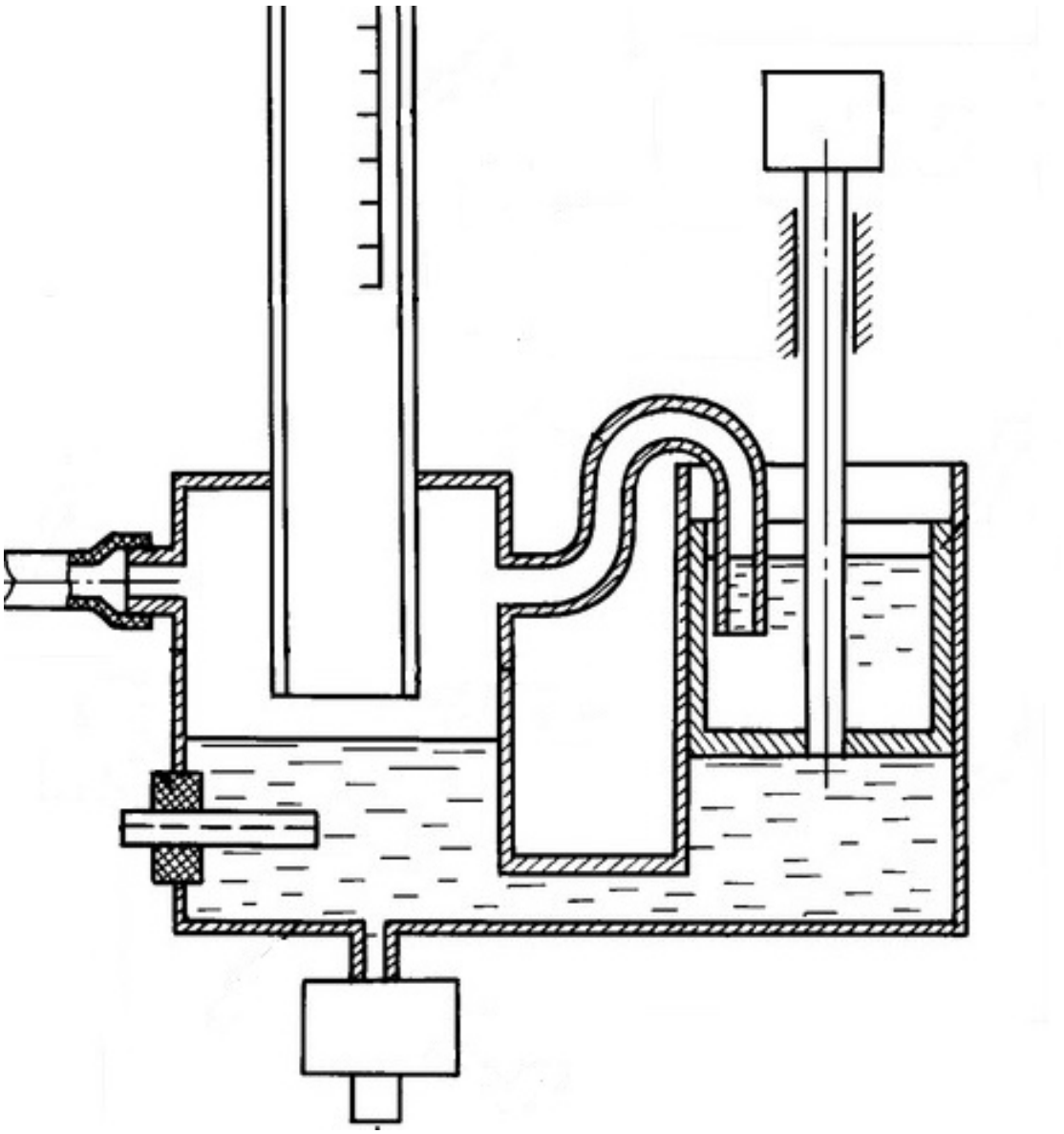


Рисунок 1.10 - Плёночный расходомер

В соответствии с рисунком 1.10, шток 6 перемещает поршень 7 с жидкостью 8, вытесняя часть раствора 4 из ёмкости 9 в резервуар 3. Уровень в резервуаре повышается и становится выше среза нижнего конца трубки 1. Газ, поступающий в резервуар 3 по газоподводящей трубке 2, выходит в атмосферу через запорную трубку 5. Затем поршень 7 поднимается в исходное положение, образовавшаяся пленка под действием газов перемещается по трубке 1. При этом расход газа пропорционален скорости перемещения плёнки по трубке 1.

Широкое распространение расходомеры такого типа получили из-за относительной простоты конструкции и как следствие малого количества погрешностей. Расход газа измеряется циклически по скорости движения пленки в бюретке. Следует заметить, что с

течением времени, принципиальные решения при разработке подобных устройств не изменялись. Изменения касались первичных преобразователей используемых для регистрации времени перемещения пленки и вторичных преобразователей, формирующих импульс длительности перемещения и измеряющих длительность этого импульса.

Измерения расхода газа пузырьковым методом, имеет достаточно большую историю. Изначально пузырьковый метод реализован в устройстве, разработанном Г. С. Бондаревым и В. С. Малышевым (6), при реализации которого, осуществляется барботаж измеряемого газа через слой жидкости с одновременным подсчетом числа пузырьков газа. При этом диаметр образующихся в системе пузырьков с газом определяется по формуле:

$$d = \sqrt[3]{\frac{6 \cdot D_C \cdot \sigma}{(\rho_Ж - \rho_Г) \cdot g}}; \qquad (1.1)$$

где Dc - диаметр сопла, м;

а - поверхностное натяжение жидкости, Н/м;

Рж - плотность жидкости, кг/м3;

р,. - плотность газа, кг/м3;

g - ускорение свободного падения, м/с2

Последующие разработки в области пузырьковых методов контроля были направлены на инженерные и технологические улучшения. Например, на рисунке 1.11 представлен оптический датчик расхода газа.

В соответствии с рисунком 1.11, газ, расход которого измеряется, поступает в кювету 1 с жидкостью через трубопровод 2 и сопло 3 в виде пузырьков и проходит через жидкость. При этом пузырьки газа в зависимости от скорости формирования уменьшают слой жидкости, которая является непрозрачной средой между воздушной трубкой 7, фотоприемником 5 и источником света 4. Вторичный преобразователь 6, подключённый к фотоприёмнику, осуществляет счет газовых пузырьков.

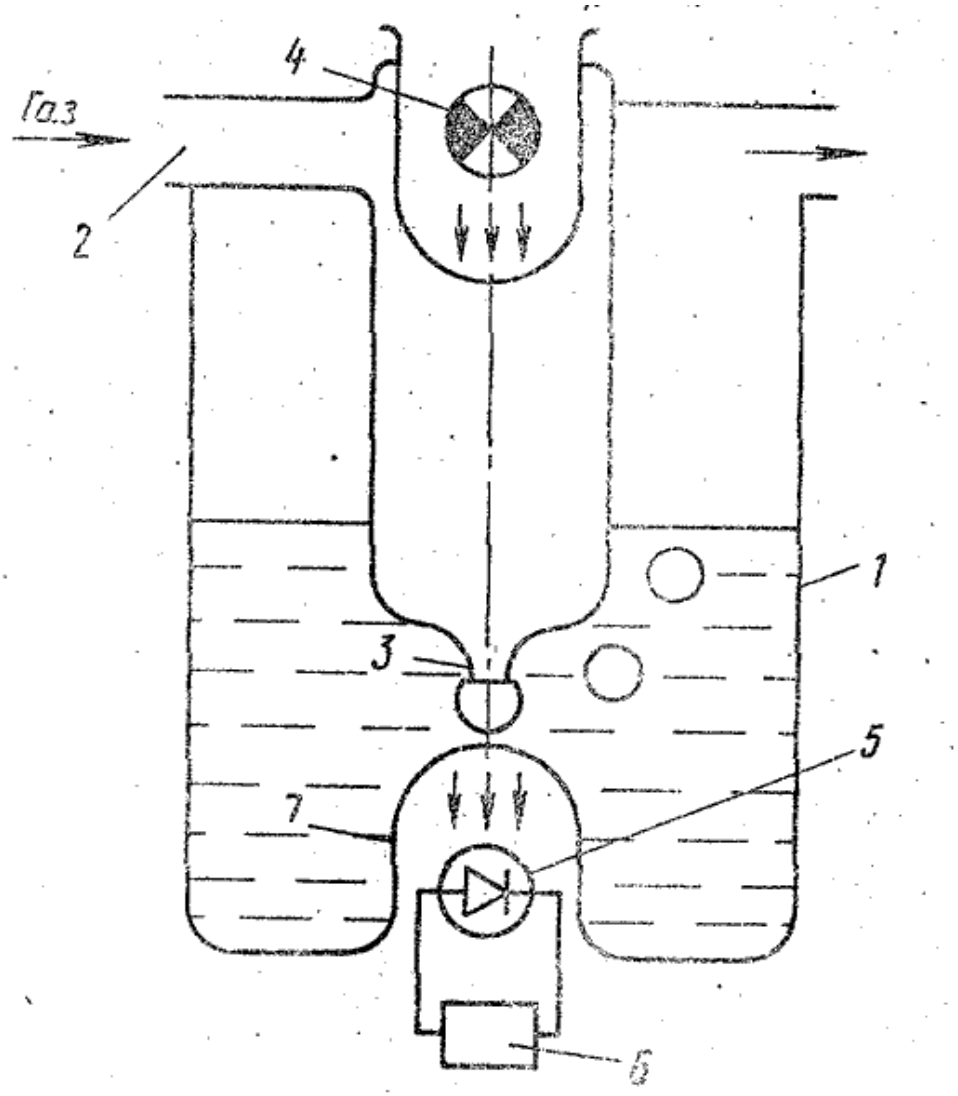


Рисунок 1.11 - Оптический датчик расхода газа

Кроме регистрации расхода газа, подобные устройства также могут регулировать соотношения малых объёмов газа для их точного дозирования при проведении исследований в экспериментальной неорганической и органической химиях, биохимии, биологии и медицине. Например, известно устройство для регистрации и регулирования соотношения малых объёмов газов, рисунок 1.12.

В соответствии с рисунком 1.12, устройство работает следующим образом. Газы поступают в корпус 1 через механизмы 8 подачи компонентов газа по впускным трубкам для газов. Образовавшиеся пузырьки заданного диаметра проходят через жидкость и лопаются в объёме над ней. В объёме над жидкостью происходит смешивание газов. Смесь газов выходит по трубке 3 для отведения смеси газов. Сигналы

с микрофонных датчиков 4, соответствующие моментам срыва пузырьков с кромки трубок 2 для газов, поступают на усилители

5. Усиленные сигналы поступают на счётные блоки 6. Необходимое число пузырьков того или иного газа, поступающего в объём над жидкостью, устанавливается исполнительными механизмами 8 автоматически по сигналам с блоков автоматического регулирования подачи газов 7.

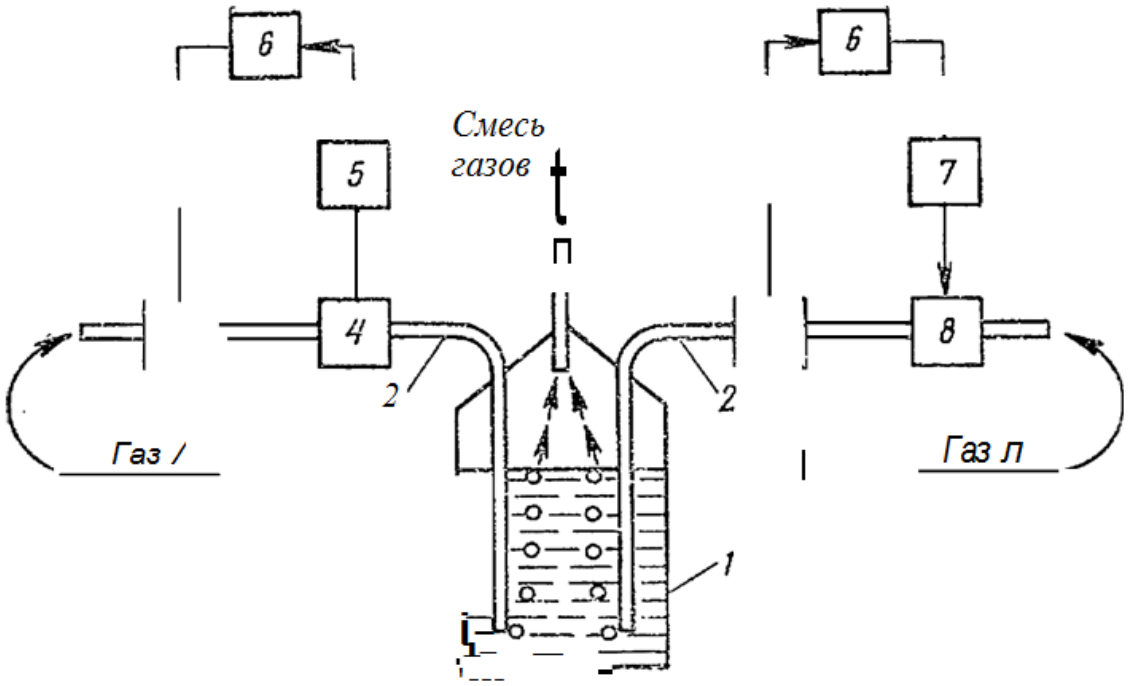


Рисунок 1.12 - Функциональная схема устройства регулирования соотношения малых объёмов газов.

Однако в подобных дискретных устройствах отсутствует учет целого ряда параметров среды, используемой для барботирования. Такие недостатки свойственны пузырьковым расходомерам. Например, в (7) рекомендуется при реализации метода, применять жидкости с высокой температурой кипения.

Там же указывается, что при правильном выборе поверхностно-активного вещества погрешность измерения расхода не будет превышать 0,7%. Подобные рекомендации приводят к увеличению времени проведения измерения и ограничивают области применения таких методов и устройств.

Кроме того, принято считать, что объем пузырька остается постоянным и определяется в первую очередь величиной диаметра сопла. Однако, в (7) утверждается, что в капельных и пузырьковых расходомерах, изменение температуры сказывается на

объёме пузырька с газом. Так, изменение температуры воды на 15 °С вызывает изменение объёма и массы на 2%. Кроме того, при реализации метода, система находится в таком режиме, что каждый пузырек газа образуется в отдельности, отрывается и поднимается независимо от других, не образуя струю. Это накладывает ограничения при проведении измерения, связанные с необходимостью термостатирования, и также снижает возможности метода и средства измерения.

Таким образом, необходимо вывести ряд зависимостей, определяющих влияние комплекса параметров на поведение пузырька с газом при осуществлении барботирования через вязкую среду.

1.3. Обзор МКМ-7

В работе объектом изучения является побудитель расхода МКМ-7 – магнитоэлектрический микрокомпрессор. Производитель – ЗАО «Эксис». МКМ-7 поставляется в виде отдельного устройства, а также как встроенный побудитель расхода в составе газоанализаторов серии МАГ, ПКУ и ПКГ. В комплект поставки отдельного устройства входит сам МКМ-7 и паспорт.

Ниже представлены паспортные данные микрокомпрессора, указываемые производителем:

1. Производительность: от 0,03 до 0,16 л/мин (не менее)
2. Питание: напряжение от 3 до 10 В, меандр частотой 400 Гц
3. Ток потребления: не менее 30 мА при напряжении питания 5 В
4. Габаритные размеры: 25*25*18 мм
5. Масса: 29 г

Также производителем представлены зависимости:

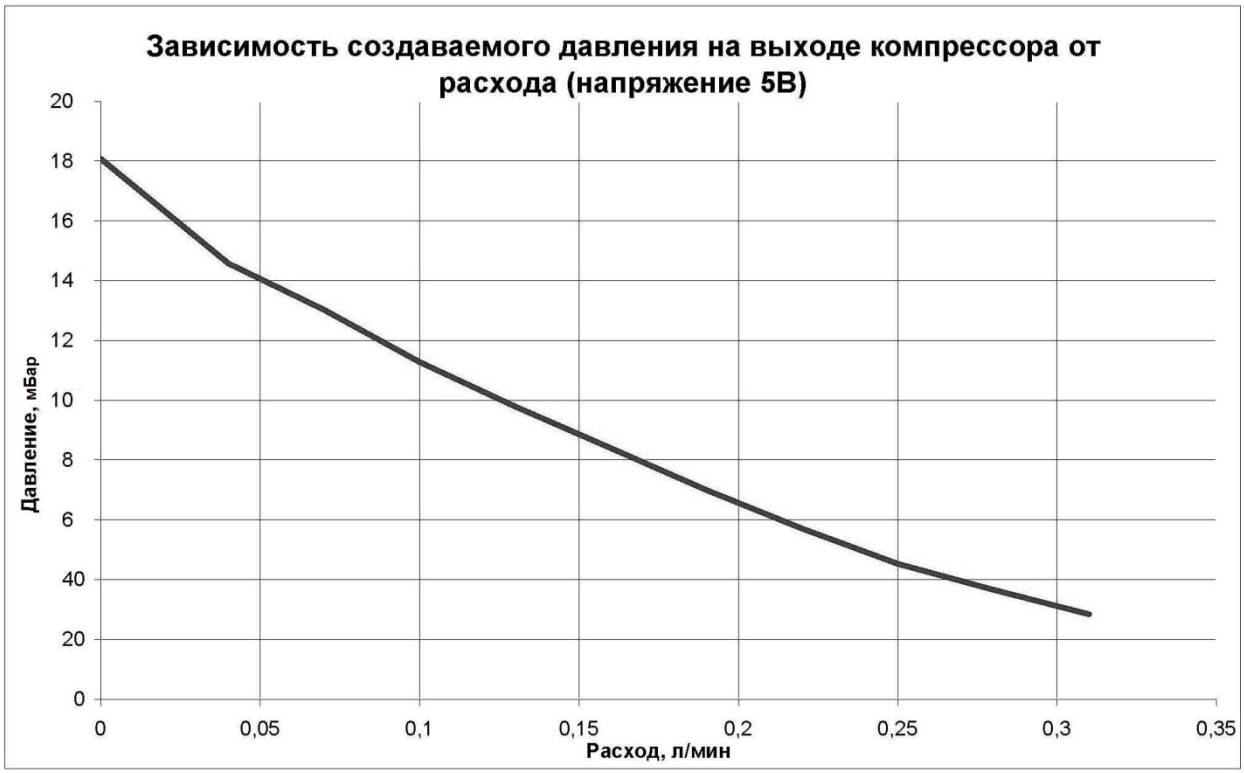


Рисунок 3. Зависимость создаваемого давления на выходе компрессора от расхода (Напряжение 5В, частота неизвестна) - данные производителя

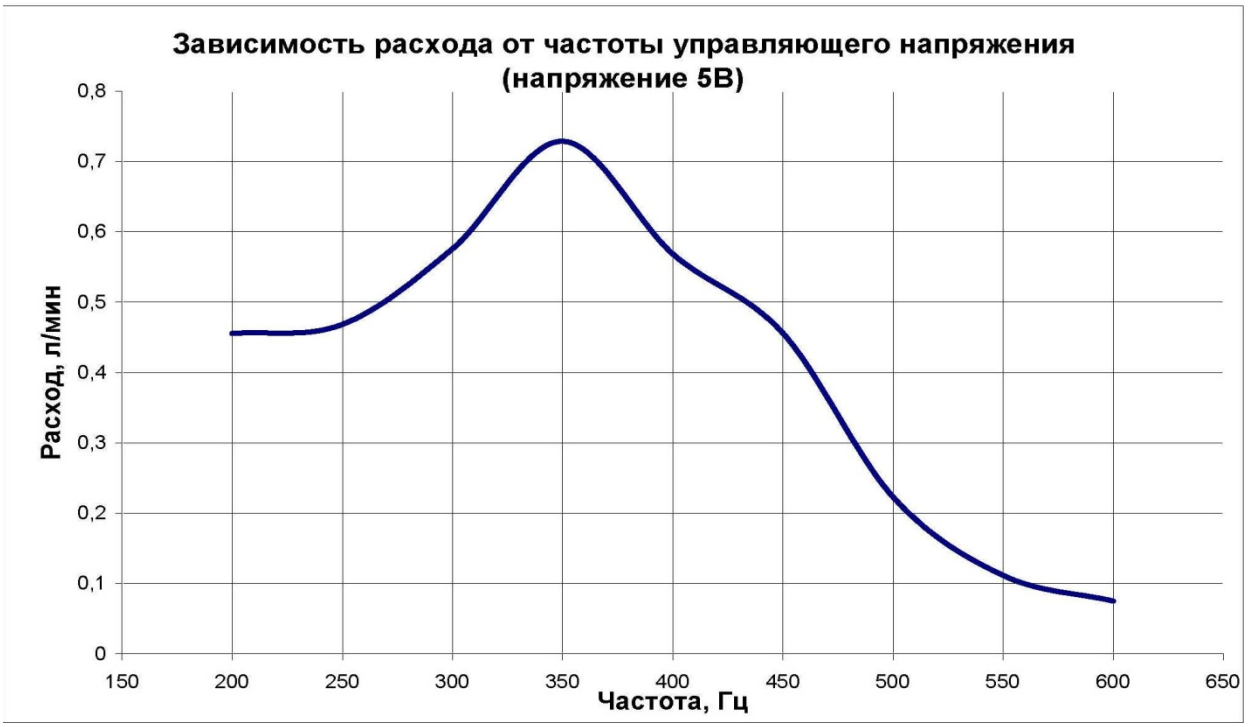


Рисунок 4. Зависимость расхода от частоты управляющего напряжения (напряжение 5В, резонанс при 350 Гц) – данные производителя.

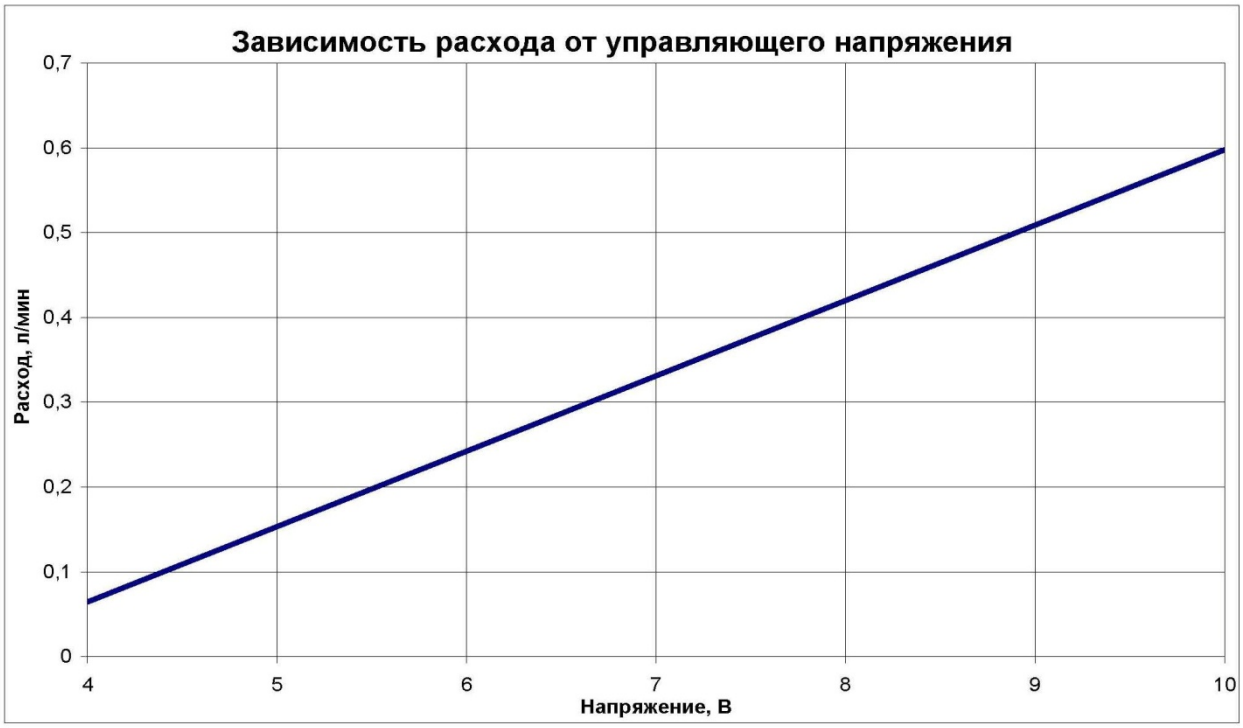


Рисунок 5. Зависимость расхода от управляющего напряжения (частота неизвестна)– данные производителя.

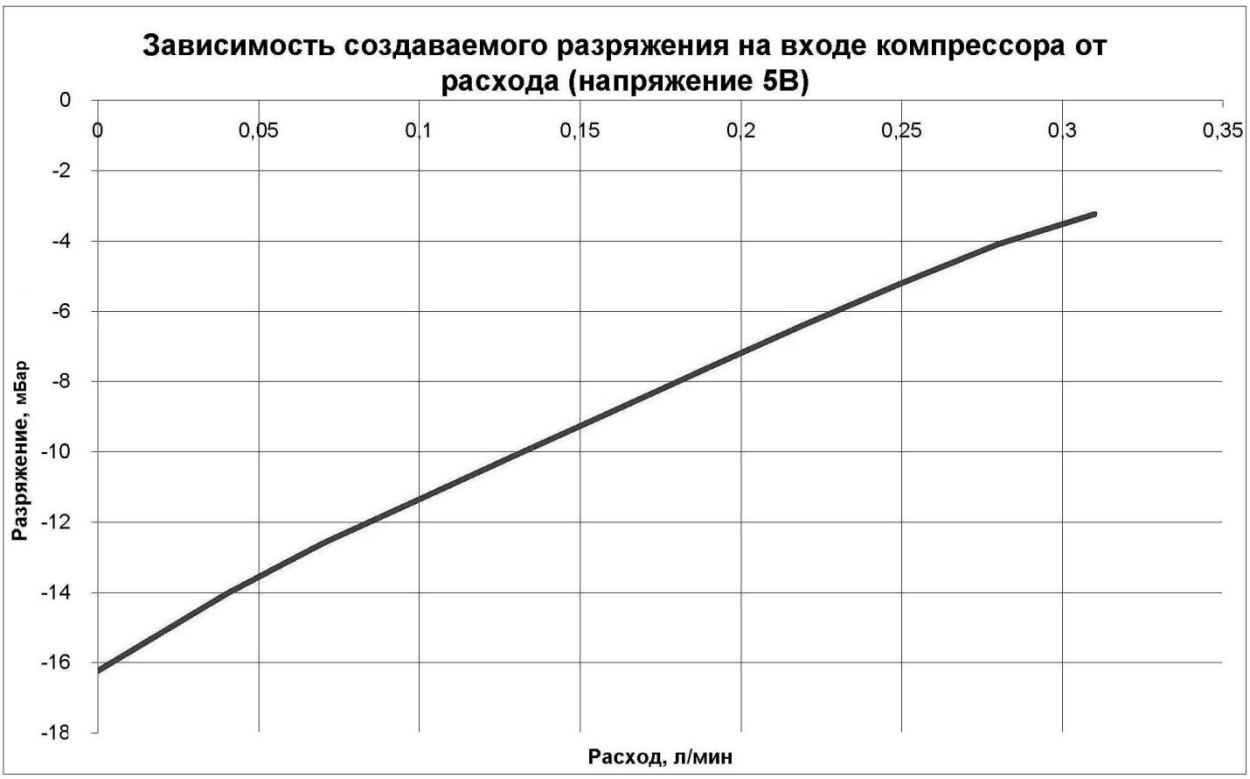


Рисунок 6. Зависимость создаваемого разряжения на входе компрессора от расхода (частота неизвестна) – данные производителя.

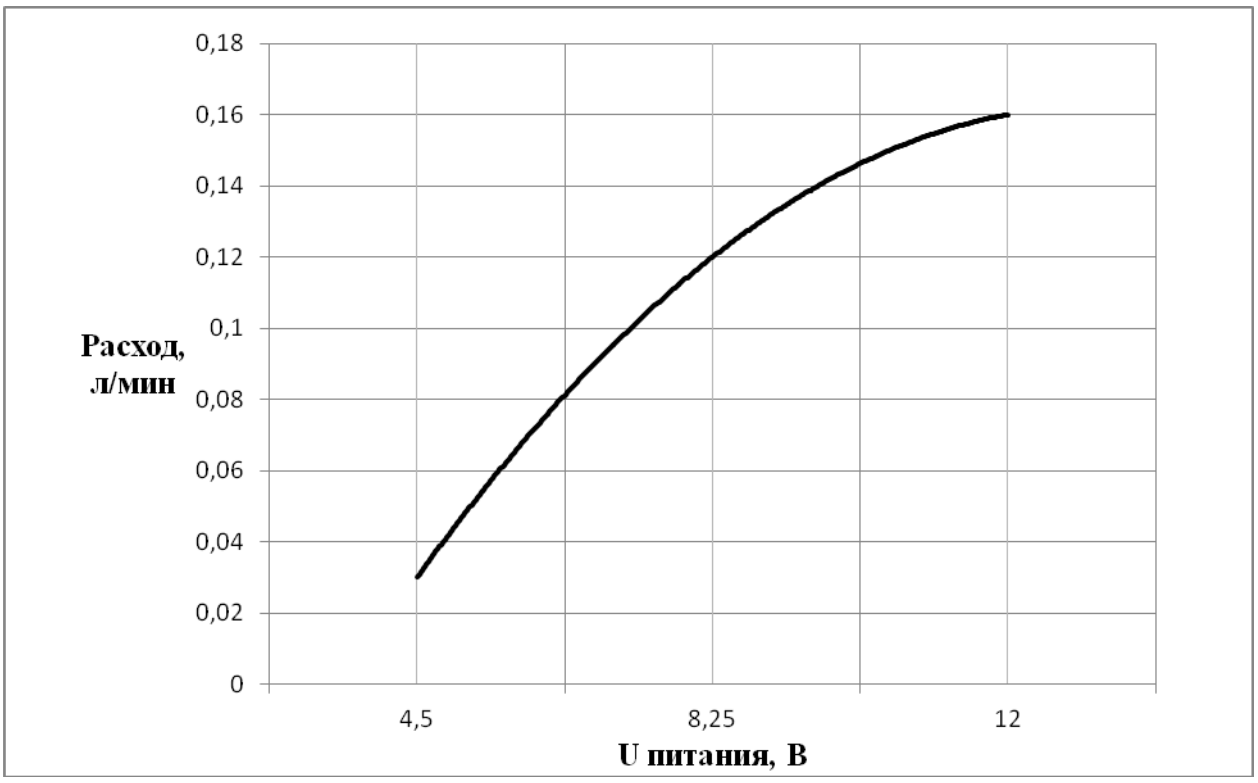


Рисунок7. Зависимость производительности от управляющего напряжения (частота неизвестна) – данные производителя.

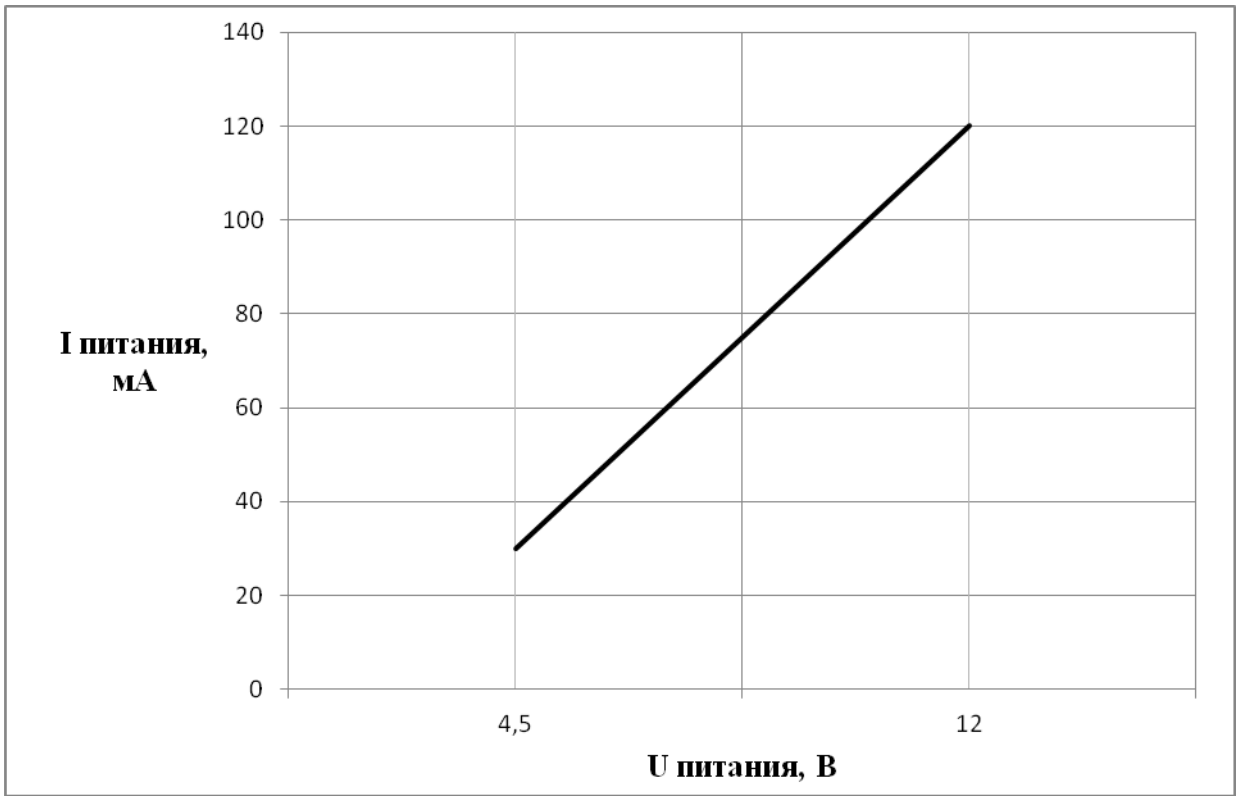


Рисунок 8. Вольт-амперная характеристика МКМ-7 (частота не указана) – данные производителя.

Рисунки 3-6 представлены на официальном сайте производителя. Рисунок7 и рисунок 8 взяты из руководства по применению МКМ-7, данного производителем. Из представленных графиков видно, что некоторые данные не указаны, а именно:

1. На Рисунке 3, 5 и 6 не указана частота управляющего напряжения
2. На Рисунке 6 не указаны данные о расходе при частоте управляющего напряжения менее 200 Гц
3. На Рисунке 5 не указаны данные о расходе при управляющем напряжении менее 4 В.
4. На Рисунках 5 и 7 представлены идентичные характеристики. Графики противоречат друг другу: на Рисунке 3 зависимость прямолинейная, на Рисунке 4– гиперболическая, также указанный расход на Рисунке 3 значительно превышает расход, указанный на Рисунке 5.

1.4. Вывод

Процессы, сопровождающиеся выделением малых объёмов газа, широко распространены в промышленных областях связанных с технологиями материалов, биологическими технологиями и инженерией.

К средствам инструментального контроля технологических процессов относят расходомеры и счётчики количества. Существующие расходомеры и счётчики количества позволяют решать задачу контроля расхода малых объёмов газа, но ограничены порогом чувствительности метода. Очевидно, что повышение порога чувствительности приведет к миниатюризации средств измерения и увеличению числа погрешностей, и как следствие – увеличение числа компенсационных цепей. Следовательно, сложность конструкции и изготовления, средства измерения расхода, будет требовать использования сложных образцовых расходомеров и разработки методов поверки, для обеспечения выходной операции контроля и градуировки.

Анализ, показал, что наиболее перспективным является пузырьковый метод контроля расхода малых объёмов газа. Это связано, с высоким порогом чувствительности, простотой конструкции и как следствие - высокой точностью измерения. Следует отметить, что в пузырьковых методах, влияние температуры на процессы измерения объемного расхода газа - велико. Очевидно, с целью повышения точности, необходимо разработать новый принцип измерения, позволяющий компенсировать влияние температуры на результат измерения.

При выполнении обзора МКМ-7 на основании анализа рабочих характеристик данный микрокомпрессор был выбран в качестве побудителя расхода для исследуемой измерительной системы.

2. Конструкторская часть

2.1. Описание метода измерения и разработка средства измерения объёмного расхода газа

Разрабатываемый метод и средство предназначены для измерения расхода и количества (объем) квазистацинарных потоков газов, у которых температура образования росы по воде не превышает температуру газа (для природного газа не выше чем по стандартам на газы (8)) и (3), с вязкостью от $6*10^{-6}$ до $35 * 10^{-6}$ Пахе, а также реологических свойств жидкости, а именно – динамической вязкости и коэффициента поверхностного натяжения.

На основании результатов исследований (см. главу 4) и (9), сформированы ограничения математической модели, и как следствие самого метода контроля расхода газа путем барботирования. Требования к измерительной системе и проведению измерения:

- форма пузырьков —сферическая (см. таблицу 3.1);
- пузырьки газа поднимаются вертикально вверх вдоль оси цилиндра;
- каждый пузырек газа образуется в отдельности, отрывается и поднимается независимо от других, не образуя струю;
- температура газа не должна превышать значений, при которых реологические свойства жидкости выходят за пределы области работоспособности барботажной системы;
- концентрация жидкости остается постоянной с течением времени;
- точность измерения зависит от поверхностного натяжения, влияющего на процесс образования пузырька у границы сопла под действием температуры.

Искомые значения (расход) определяются с помощью информационно - вычислительного комплекса (далее - ИВК). ИВК - совокупность средств измерений и вспомогательных устройств, объединенных в единую функциональную систему, обеспечивающую измерение количества (расход) газа в рабочих условиях, определение параметров газа, приведение измеренного количества к стандартным условиям.

В общем случае в состав ИВК входят:

- цилиндрическая кювета, заполненная вязкой жидкостью, с соплом, расположенном в основании цилиндра;

- трубопровод для подачи и вывода газа, расход которого измеряется;

- видеорегистрирующее устройство, подключенное к средству автоматической обработки информации ЭВМ.

Диаметр сопла определяется в соответствии рекомендациями

Диаметр кюветы D_K выбирается из условия:

$$D_K > 10 \cdot D_S . \qquad (2.1)$$

Рассмотрим конструктивные особенности измерительной системы связанные с уровнями расположения оптических линий регистрации информации о процессе перемещения пузырька.

2.3. Вывод

Предлагаемый метод измерения расхода малых объёмов газа позволяет учесть изменения реологических параметров барботажной жидкости, влияющие на величину объёма пузырька газа.

В предлагаемом аппаратно-программном комплексе контроля расхода малых объёмов газа, жидкость для барботирования должна обладать следующими показателями: динамическая вязкость не менее 25 Па·с, коэффициент поверхностного натяжения должен находиться в пределе от 60,7 до 63,4 Н/м

3. Технологическая часть

3.1. Рекомендации к проведению измерений

В соответствии с (8) измерения выполняют при следующих условиях:

климатические условия эксплуатации применяемых средств измерений должны соответствовать требованиям, установленным в технической документации на них;

диапазоны изменений параметров, измеряемых применяемыми средствами измерений, должны находиться в диапазонах применения этих средств с нормированными метрологическими характеристиками;

параметры энергопитания средств измерений должны находиться в пределах, нормированных в их технической документации;

все средства измерений должны иметь свидетельства о поверке или оттиски поверительных клейм;

все средства измерений и вспомогательные устройства должны применяться в соответствии с требованиями действующих для них нормативных и руководящих документов по технической эксплуатации и безопасности;

к выполнению измерений допускают лиц, изучивших соответствующие инструкции по технике безопасности.

Учитывая расчётные соотношения, разработан общий алгоритм измерения, основанный на выполнении следующих пунктов:

предварительно должны быть установлены следующие технические характеристики: диаметр сопла, расстояние между цилиндрической кюветой и видеокамерой, плотность используемого газа при нормальных условиях;

требования к видеокамере: не менее 30 fps (кадров в секунду)

кювету заполняют исследуемой жидкостью (в случае возникновения в ней воздушных пузырьков, выжидают до их самостоятельного удаления, а в случае их прилипания к стенке с помощью вибрации или механическим путём);

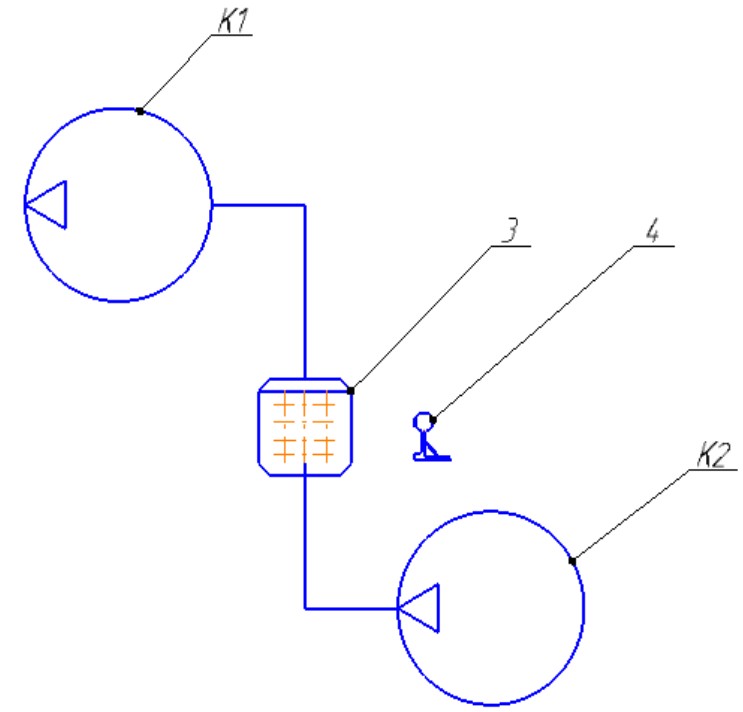
в сопло от внешнего источника подают газ обеспечивая отрыв от сопла в жидкость одного пузырька;

определяют динамическую вязкость жидкости;

при барботировании пузырька через выбранную жидкость выполняется вычисление значения объёма газа в пузырьке путём численного решения уравнения (4.41) и запоминание полученного значения;

значение объёмного расхода газа вычисляется по соотношению (4.41).

3.2. Описание испытательного стенда



K1 – компрессор понижения давления

K2 – нагнетающий компрессор

3– измерительная кювета

4– измерительная видеокамера

Рис. 3.1 – испытательный стенд

3.3. Конструирование программной части

Первый важный этап выделения контуров - преобразование изображения в серые тона:

```
float red = (pixels[i + j * w] >> 16) & 0xff;
 float green = (pixels[i + j * w] >> 8) & 0xff;
 float blue = (pixels[i + j * w]) & 0xff;
 int gray = (int) (red * .3 + green * .59 + blue * .11);
```

Остальные этапы схематично представлены на рисунке ниже:

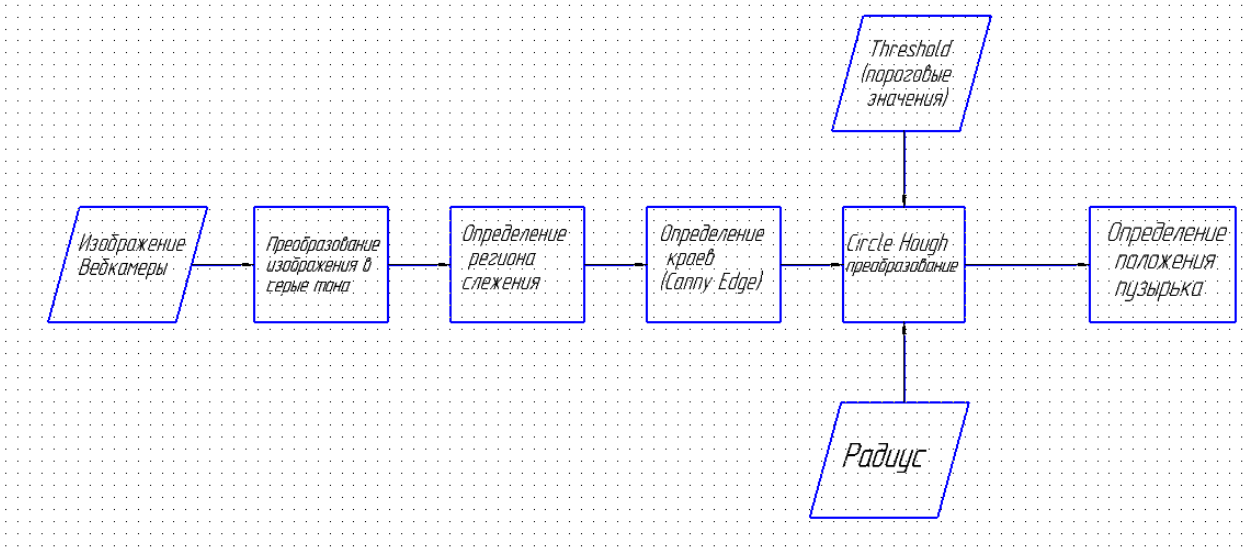


Рисунок 3.2 Схема распознования сферических объектов (пузырьков газа)

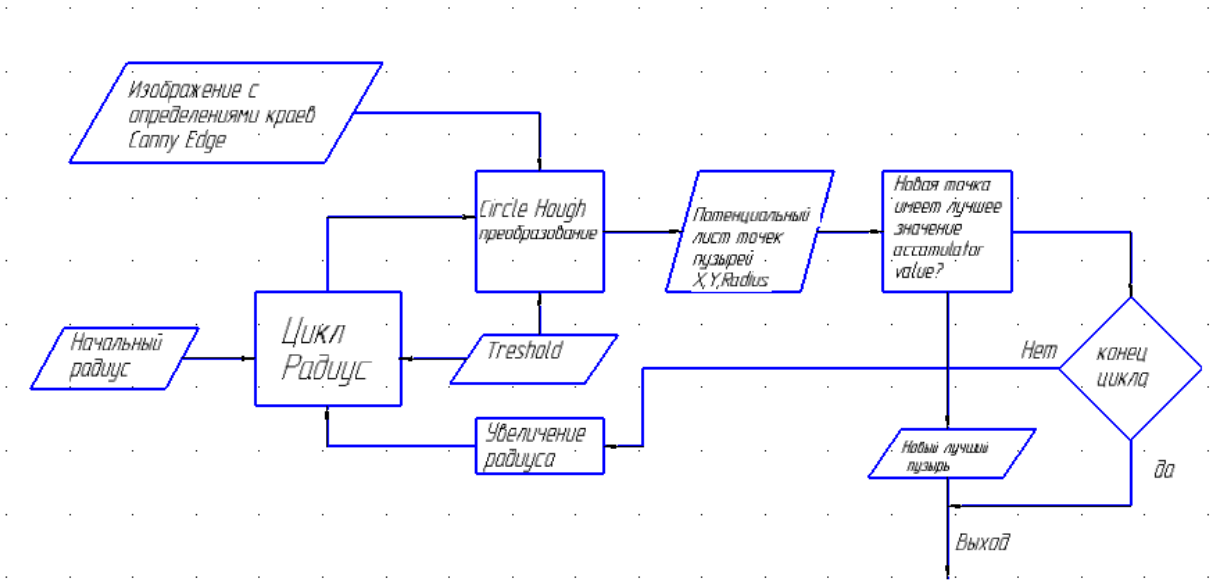


Рисунок 3.2 Схема цикла выделения наилучшего сферического объекта (пузырьков газа) из возможных.

3.4. Методика проведения эксперимента

В соответствии с (8) измерения выполняют при следующих условиях:

климатические условия эксплуатации применяемых средств измерений должны соответствовать требованиям, установленным в технической документации на них;

диапазоны изменений параметров, измеряемых применяемыми средствами измерений, должны находиться в диапазонах применения этих средств с нормированными метрологическими характеристиками;

параметры энергопитания средств измерений должны находиться в пределах, нормированных в их технической документации;

все средства измерений должны иметь свидетельства о поверке или оттиски поверительных клейм;

все средства измерений и вспомогательные устройства должны применяться в соответствии с требованиями действующих для них нормативных и руководящих документов по технической эксплуатации и безопасности;

к выполнению измерений допускают лиц, изучивших соответствующие инструкции по технике безопасности.

Учитывая расчётные соотношения, разработан общий алгоритм измерения, основанный на выполнении следующих пунктов:

1. Предварительно должны быть установлены следующие технические характеристики: диаметр сопла, расстояние между цилиндрической кюветой и видеокамерой, плотность используемого газа при нормальных условиях;
2. Проверяется работоспособность датчика давления
3. Жидкостная ёмкость дифференциального манометра заливается техническим этиловым спиртом плотностью $0{,}8095\,\frac{\text{кг}}{\text{м}^3}$, выставляется нулевой уровень, тумблер выставляется в режим положительного или отрицательного перепада давления относительно атмосферного в зависимости от режима испытания.
4. Проводится калибровка датчика давления в соответствии с руководством (10) .
5. С помощью регуляторов амплитуды и частоты управляющего напряжения устанавливается заданный режим питания катушки МКМ-7. Сигнал на катушке считывается АЦП и передаётся на ЭВМ.
6. Проводится калибровка видеокамеры.
7. С помощью сбрасывающего вентиля в ресивере устанавливается атмосферное давление.
8. Кювету заполняют исследуемой жидкостью (в случае возникновения в ней воздушных пузырьков, выжидают до их самостоятельного удаления, а в случае их прилипания к стенке с помощью вибрации или механическим путём);
9. Запускается МКМ-7. В сопло подается газ, обеспечивая отрыв от сопла в жидкость одного пузырька;

10. Определяют динамическую вязкость жидкости с помощью пузырька-маркера.
11. Вычисляется значение объёма газа в пузырьке путём численного решения уравнения (4.21) и запоминание полученного значения;
12. Вычисляется объёмный расход газа по соотношению (4.22).

3.5. Обработка полученных данных

Проведём синтез статической характеристики для измерительной кюветы барботажной системы.

С этой целью, следует определить функциональную зависимость между диаметром пузырька (являющимся выходным параметром, который подлежит контролю) и температурой (являющаяся внешним, входным фактором). Для этого, определим ряд выражений.

Зависимость между плотностью газа и температурой определяется по формуле:

$$P_г == P_{оа} \cdot (1 + ag \cdot T), \qquad (3.1)$$

где $P_{оа}$ - плотность газа при и.у.;

ag - коэффициент объёмного расширения газа;

T – входной параметр изменения температуры.

Зависимость между плотностью жидкости и температурой определяется по формуле:

$$p_ж = P_{oQ} \cdot (1 + ag \cdot T) \qquad (3.2)$$

где P_{oQ} - плотность газа при и.у.;

ag - коэффициент объёмного расширения газа.

Объём газа внутри сферического пузырька определим по формуле:

$$V_Г = \frac{\pi \cdot D_{PUZ}^{3}}{6}. \qquad (3.3)$$

С учётом (4.20), перепишем относительно диаметра пузырька D_{puz}, получим:

Из формулы (3.15) видно, что коэффициент поверхностного натяжения есть функция от динамической вязкости. Из различных видов математической обработки данных был выбран квадратичный полином и получено уравнение:

Предварительное описание и распределение частных погрешностей приведено в таблице 3.1.

Источник	Первичная погрешность	Характеристики погрешности
1	2	3
Неидеальности изготовления элементов измерительной кюветы	Погрешность от несоответствия диаметра сопла номинальному значению ΔD_S	$\Delta D_S = +6 \cdot 10^{-6}$ м; $M(\gamma(D_S)) = E_0(D_S) + \alpha_q \cdot T_1(D_S) = 3 \cdot 10^{-6}$ м; $S(\gamma(D_S)) = \lambda \cdot T_1(D_S) = 0{,}9 \cdot 10^{-6}$ м.
	Погрешность от несоответствия внутреннего диаметра измерительной кюветы номинальному значению ΔD_K	$\Delta D_K = {}^{+0{,}010}_{-0{,}005}$ мм; $M(\gamma(D_K)) = E_0(D_K) + \alpha_q \cdot T_1(D_K) = 4{,}8 \cdot 10^{-6}$ м; $S(\gamma(D_K)) = \lambda \cdot T_1(D_K) = 3 \cdot 10^{-6}$ м.
	Погрешность от несоответствия глубины залегания сопла ΔH	$\Delta H = \pm 50 \cdot 10^{-6}$ м; $M(H) = 0$; $S(H) = \Delta H / t_\alpha = 1.667 \cdot 10^{-5}$;
	Погрешность от несоответствия расстояния между видеокамерой и измерительной кюветой	$\Delta l = \pm 50 \cdot 10^{-6}$ м; $M(l) = 0$; $S(l) = \Delta l / t_\alpha = 1.667 \cdot 10^{-5}$;
Погрешности временных задержек	Погрешность дискретизации при преобразовании видеосигнала	$\Delta t = 60$ мс (при fps=15) $M(\Delta t) = \Delta t / 2$ $S(\Delta t) = \Delta t / t\alpha$; $t\alpha = 3$;

1	2	3
Неидеальность параметров АЦП	Погрешность квантования $\Delta_{КВ}$	$\Delta_{КВ} = U_{АЦПМАХ} / 2^m$, где: m – число разрядов АЦП.
	Погрешность $\Delta_Л$ линейности АЦП	$M(\Delta_Л) = \Delta_Л / 2$; $S(\Delta_Л) = \left(\Delta_Л / 2\right) / t_\alpha$; $t_\alpha = 3$.
	Погрешности $\Delta_{ЛД}$ от дифференциальной нелинейности АЦП	$M(\Delta_{ЛД}) = 0$; $S(\Delta_Л) = \Delta_{ЛД} / t_\alpha$; $t_\alpha = 3$.

Таблица 3.3 - Предварительное распределение частных погрешностей

Оценки первичных погрешностей	Коэффициенты влияния	Оценки частных погрешностей
$\gamma Ds = 1$; $M(\gamma D_S(T)) = 0$; $S(\gamma D_S(T)) = 0.577$;	$KDs = 2.188$;	$M(\Delta D_{PUZ}(T)) = \left(\frac{dD_{PUZ}}{dD_S}\right)_0 \cdot M(\gamma D_S(T)) = 0$; $S(\Delta D_{PUZ}(T)) = \left\lvert\frac{dD_{PUZ}}{dD_S}\right\rvert_0 \cdot S(\gamma D_S(T)) = 1.262$;
$\gamma Dk = 1$; $M(\gamma D_K(T)) = 0$; $S(\gamma D_K(T)) = 0.577$;	$KDk = -1.21$;	$M(\Delta D_{PUZ}(T)) = \left(\frac{dD_{PUZ}}{dD_K}\right)_0 \cdot M(\gamma D_K(T)) = 0$; $S(\Delta D_{PUZ}(T)) = \left\lvert\frac{dD_{PUZ}}{dD_K}\right\rvert_0 \cdot S(\gamma D_K(T)) = 0.698$;
$\gamma l(T) = 1$; $M(\gamma l(T)) = 0$; $S(\gamma l(T)) = 0.577$;	$Kl = 0$;	$M(\Delta D_{PUZ}(T)) = \left(\frac{dD_{PUZ}}{dl}\right)_0 \cdot M(\gamma l(T)) = 0$; $S(\Delta D_{PUZ}(T)) = \left\lvert\frac{dD_{PUZ}}{dl}\right\rvert_0 \cdot S(\gamma l(T)) = 0$;
$\gamma H(T) = 1$; $M(\gamma H(T)) = 0$; $S(\gamma H(T)) = 0.577$;	$KH = 9.1 \cdot 10^{-5}$;	$M(\Delta D_{PUZ}(T)) = \left(\frac{dD_{PUZ}}{dH}\right)_0 \cdot M(\gamma H(T)) = 0$; $S(\Delta D_{PUZ}(T)) = \left\lvert\frac{dD_{PUZ}}{dH}\right\rvert_0 \cdot S(\gamma H(T)) = 5.251 \cdot 10^{-5}$;

$\Delta t = 6 \cdot 10^{-2}$; $M(\Delta t) = 3{,}0 \cdot 10^{-3}$; $S(\Delta t) = 3.333 \cdot 10^{-4}$;	$Kt = -3.432 \cdot 10^{-5}$;	$M(\Delta D_{PUZ}(T)) = \left(\frac{dD_{PUZ}}{dt}\right)_0 \cdot M(\gamma(T)) = -1.03 \cdot 10^{-7}$; $S(\Delta D_{PUZ}(T)) = \left\|\frac{dD_{PUZ}}{dt}\right\|_0 \cdot S(\gamma(T)) = 1.144 \cdot 10^{-8}$;

В соответствии с математическим описанием статистических оценок первичных и частных погрешностей выполнен расчёт точности разработанного устройства.

будем предполагать, что:

1) грубые погрешности исключены;

2) поправки, которые следовало определить (например, смещение нулевого деления шкалы), вычислены и внесены в окончательные результаты;

3) все систематические погрешности известны (с точностью до знака).

В этом случае результаты измерений оказываются все же не свободными от случайных погрешностей. Если случайная погрешность окажется меньше систематической, то, очевидно, нет смысла пытаться уменьшить величину случайной погрешности - все равно результаты измерений не станут значительно лучше и, желая получить большую точность, нужно искать пути к уменьшению систематической погрешности. Наоборот, если случайная погрешность больше систематической, то именно случайную погрешность нужно уменьшить в первую очередь и добиться того, чтобы случайная погрешность стала меньше систематической, с тем чтобы последняя опять определяла окончательную погрешность результата. На практике обычно уменьшают случайную погрешность до тех пор, пока она не станет сравнимой по величине с систематической погрешностью. В целом случайная погрешность уменьшается при увеличении числа измерений.

Поскольку из-за наличия случайных погрешностей результаты измерений по своей природе представляют собой тоже случайные величины, истинного значения $x_{ист}$ измеряемой величины указать нельзя. Однако можно установить некоторый интервал значений измеряемой величины вблизи полученного в результате измерений значения $x_{изм}$, в котором с определенной вероятностью содержится $x_{ист}$. Тогда результат измерений можно представить в следующем виде:

$$x_{изм} - \Delta x \le x_{ист} \le x_{изм} + \Delta x \qquad (3.4)$$

где Δx-погрешность измерений. Вследствие случайного характера погрешности точно определить ее величину невозможно. В противном случае найденную погрешность можно было бы ввести в результат измерения в качестве поправки и получить истинное значение $x_{ист.}$. Задача наилучшей оценки значения $x_{ист}$ и определения пределов интервала (11) (12) (13) по результатам измерений является предметом математической статистики. Воспользуемся некоторыми ее результатами.

Пусть проведено n измерений величины x. Тогда за лучшую оценку истинного значения результата измерений принимается среднее арифметическое значение

$$\langle x \rangle = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad (3.5)$$

где x_i - результат i -го измерения.

Для оценки случайной погрешности измерения существует несколько способов. Наиболее распространена оценка с помощью стандартной или средней квадратичной погрешности σ (ее часто называют стандартной погрешностью или стандартом измерений).

Средней квадратичной погрешностью называется величина:

$$S_n = \sqrt{\frac{\sum_{i=1}^{n} \left(\langle x \rangle - x_i\right)^2}{n-1}} \qquad (3.6)$$

где n – число наблюдений.

Если число наблюдений очень велико, то подверженная случайным колебаниям величина S_n стремится к постоянному значению σ:

$$\sigma = \lim_{n \to \infty} S_n \,. \qquad (3.7)$$

Именно этот предел и входит в качестве параметра в распределение Гаусса. (12)

Квадрат этой величины называется дисперсией измерений.

Пусть α означает вероятность того, что результат измерений отличается от истинного на величину, не большую, чем Δx. Вероятность α в этом случае носит название доверительной вероятности, а интервал значений измеряемой величины от $\langle x \rangle - \Delta x$ до $\langle x \rangle + \Delta x$ называется доверительным интервалом.

Определим доверительный интервал. Чем большим будет установлен этот интервал, тем с большей вероятностью $x_{ист}$ попадает в этот интервал. С другой стороны, более широкий интервал дает меньшую информацию относительно величины $x_{ист.}$ Если ограничиться учетом только случайных погрешностей, то при небольшом числе измерений n для уровня доверительной вероятности α полуширина доверительного интервала (2) равна:

$$\Delta x_{сл} = t_{\alpha,n} S \qquad (3.8)$$

Примем t_α - для доверительной вероятности 0,95 равным 2

Для определения суммарной погрешности использовано выражение:

$$\Delta_{\sum} = \sum M(\Delta N)_{qi} + t_\alpha \cdot \sqrt{\sum \left(S(\Delta N)_{qi}\right)^2}, \qquad (3.9)$$

где $M(\Delta N)_{qt}$ - математическое ожидание частной погрешности;

$S(\Delta N)_{qt}$ – среднее квадратическое отклонение частной погрешности

t_α - квантиль распределения

Из таблицы 3.3 по формуле 3.9. получаем $\Delta_{\sum} = 1{,}96\%$

3.6. Вывод

Проведённые исследования распространяются на измерение расхода и количество (объем) газа. Они устанавливают методику выполнения измерений с помощью процесса барботирования, путём контроля реологических свойств жидкости, используемой для барботажа. Разработанный метод измерения расхода малых объёмов газа позволяет учесть изменения реологических параметров барботажной жидкости, влияющие на величину объёма пузырька газа.

Исходя из анализа существующих данных приведенная погрешность не должна превышать 1,8%.

Проведенный анализ точности для измерительной кюветы и разработанного средства контроля расхода сверхмалых объёмов газа показал, приведенная суммарная

погрешность, с доверительной вероятностью 0,95, в диапазоне температур от 20 до 40 °С должна быть в пределе двух процентов.

4. Исследовательская часть

4.1 Теоретическое обоснование и принципы построение дискретного метода измерения расхода газа

4.1.1 Математическое моделирование теплообмена между газом и жидкостью в процессе барботажа

Если газ в пузырьке имеет температуру отличную от температуры вязкой жидкости, то имеет место теплообмен между газом и жидкостью (двухфазный теплообмен). При этом, если температура газа больше температуры жидкости, имеет место процесс остывания газа при движении пузырька вдоль оси цилиндра.

Таким образом, имеет место нестационарными процессами теплопроводности, при которых перенос теплоты осуществляется за счет теплопроводности с течением времени (14). При этом пузырёк газа стремится к тепловому равновесию.

Зададимся температурой окружающей средь, а именно температурой жидкости, $Тж = const$, и законом теплообмена между поверхностью тела и окружающей средой. Из источника (14) известно, что дифференциальное уравнение теплопроводности при отсутствие внутренних источников тепла имеет вид:

$$\frac{\partial T}{\partial \tau} = a\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2}\right),$$

где a - коэффициент теплопроводности (одинаков для всех точек шара, и не изменяется во времени);

$\partial x, \partial y, \partial z$ - стороны элементарного параллелепипеда;

T - температура;

τ - время.

В первом приближении форму пузырька примем сферической.

Отсчет температуры шара в любой момент времени будем вести от температуры окружающей среды:

$v = T - Tж$,

где: v - избыточная температура для любой точки шара.

В сферических координатах дифференциальное уравнение перепишем следующим образом:

$$\frac{\partial v}{\partial \tau} = a\left(\frac{\partial^2 v}{\partial r^2} + \frac{2}{r}\cdot\frac{\partial v}{\partial r}\right), \qquad (4.2)$$

где: r - радиус сферы.

Из источника (14) известно, что граничные условия третьего рода:

$$\left(\frac{\partial T}{\partial n}\right)_{n=0} = -\frac{a}{\lambda}\left(T_{n=0} - T_{ж}\right),$$

где: λ - коэффициент теплопередачи;

∂T - производная температура по нормали n

∂n

Таким образом, граничные условия, для нашего случая, окончательно запишем:

- на поверхности шара, $r = r_0$: $\left(\frac{\partial v}{\partial r}\right)_{r=r_0} = -\frac{a}{\lambda}\cdot v_{r=r_0}$;

- в центре шара, $r = 0$: $\left(\frac{\partial v}{\partial r}\right)_{r=0} = 0$.

Очевидно, решение дифференциального уравнения (4.2) с учетом начальных и граничных условий даст искомое распределение температуры.

При решении задачи, используется математическая модель, изложенная в (14). При этом используются следующие подходы и методы:

метод разделения переменных, сущность которого изложена в (15), (16);

рекомендации по выбору постоянной из граничных условий и из физических соображений (15), (16);

метод интегрирования системы обыкновенных дифференциальных уравнений (15), (16); с учётом уравнения Бесселя (17)

способ графического решения трансцендентных уравнений (15);. Решение дифференциального уравнения (4.2), с учётом изложенных подходов и методов, позволяет получить значения корней уравнения, при этом, каждому корню будет соответствовать своё частное распределение температуры в виде суммы бесконечного ряда.

При этом, полученные частные решения будут удовлетворять дифференциальному уравнению при любых значениях постоянных интегрирования и воспроизводить любую действительную температурную зависимость в начальный момент времени. Так как отдельные распределения удовлетворяют дифференциальному уравнению (4.2) и граничным условиям, то и сумма их также удовлетворяет тем же условиям.

Окончательное выражение для температурного поля при охлаждении однородного шара:

$$\nu = \sum_{n=1}^{\infty} \nu_0 \cdot \frac{2 \cdot (\sin \mu_n - \mu_n \cdot \cos \mu_n)}{(\mu_n - \sin \mu_n \cdot \cos \mu_n)} \cdot \frac{\sin(\mu_n \cdot R)}{\mu_n \cdot R} \cdot \exp\left(-\mu_n{}^2 \cdot \frac{a \cdot \tau}{R^2}\right) \qquad (4.3)$$

Поскольку, отсчет температуры шара в любой момент времени будем вести от температуры окружающей среды, то избыточная температура для любой точки шара, $\frac{\nu}{\nu_0} = \theta|$, перепишем (4.3):

$$\theta = \sum_{n=1}^{\infty} \frac{2 \cdot (\sin \mu_n - \mu_n \cdot \cos \mu_n)}{(\mu_n - \sin \mu_n \cdot \cos \mu_n)} \cdot \frac{\sin(\mu_n \cdot R)}{\mu_n \cdot R} \cdot \exp\left(-\mu_n{}^2 \cdot \frac{a \cdot \tau}{R^2}\right) \qquad (4.4)$$

, где μ_n – значение корня характеристического уравнения

R – радиус шара

Из анализа уравнения (4.4) следует, что при значениях $\frac{a \cdot \tau}{R^2} >= 0{,}25$

ряд становится настолько быстро сходящимся, что для выражения температурного поля можно ограничиться первым членом ряда:

$$\theta = \frac{2 \cdot (\sin \mu_1 - \mu_1 \cdot \cos \mu_1)}{(\mu_1 - \sin \mu_1 \cdot \cos \mu_1)} \cdot \frac{\sin(\mu_1 \cdot R)}{\mu_1 \cdot R} \cdot \exp\left(-\mu_1^2 \cdot \frac{a \cdot \tau}{R^2}\right) \qquad (4.5)$$

Графически, процесс изменения температуры сферического объекта (остывание) с течением времени, изображён на рисунке 4.1.

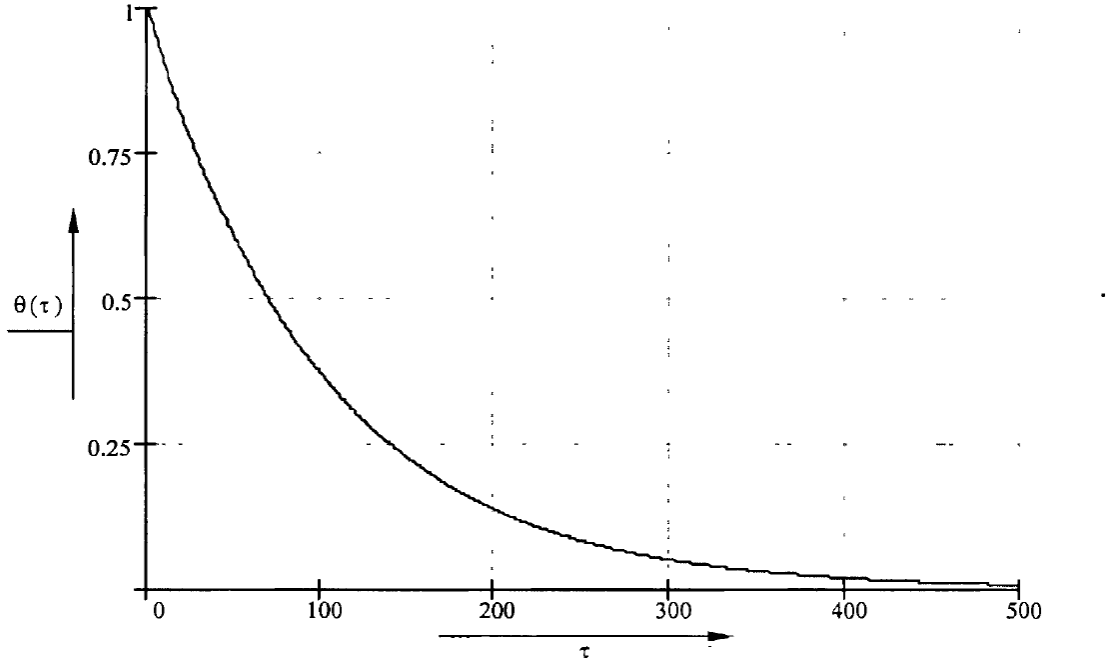


Рисунок 2.1 - Изменение температуры сферического пузырька с течением времени

Таким образом, формируется распределение изменения температуры сферического пузырька, проходящего вдоль оси цилиндра, в зависимости от времени, которое он затрачивает на движение по траектории.

Очевидно, целесообразно определить распределение температуры по слоям однородного цилиндра жидкой среды. При этом тепло передается из области с более высокой температурой в область с более низкой температурой за счет процесса теплопроводности. Источником тепла, а нашем случае, будет трек нагретых точек образующих ось цилиндра, оставленных после прохождения сферического пузырька.

Из источника (14) известно, что тепловой поток Q, направленный по нормали к площади поперечного сечения F, под воздействием разности температур ΔT между двумя точками, разделенными расстоянием L, выражается следующим соотношением:

$$Q = \lambda \cdot \frac{F \cdot \Delta T}{L}. \qquad (4.6)$$

Плотность теплового потока в данной точке, когда F стремится к бесконечно малому значению, определяется соотношением:

$$\frac{dQ}{dS} = -\lambda \cdot \frac{dT}{dR} \qquad (4.8)$$

Знак минус указывает на то, что тепловой поток распространяется в направлении убывания температуры, т. е. на отрезке ΔL температура уменьшается на ΔT .

Перепишем для сферы площадью S и радиуса R:

$$\frac{Q}{4 \cdot \pi \cdot R^2} = -\lambda \cdot \frac{dT}{dR}. \qquad (4.9)$$

Преобразуем левую часть выражения (4.8), и интегрируя получим:

$$\frac{Q}{4 \cdot \pi} \cdot \left(-\frac{1}{R} \right)_{R1}^{R2} = -\lambda \cdot T_{t1}^{t2}.$$

Таким образом, распределение температуры для полой сферы будет описываться выражением:

$$\frac{Q}{4 \cdot \pi} \cdot \left(\frac{1}{R_2} - \frac{1}{R_1} \right) = -\lambda \cdot (T_2 - T_1). \qquad (4.11)$$

Определим распределение температуры, для этого, выберем точку между R 1 и R2

$$\frac{4\cdot\pi\cdot\lambda\cdot(T_1-T)}{\frac{1}{r_1}-\frac{1}{r}}=\frac{4\cdot\pi\cdot\lambda\cdot(T-T_2)}{\frac{1}{r}-\frac{1}{r_2}} \qquad (4.12)$$

Преобразуем выражение (4.12), получим:

$$\frac{T_1-T}{T-T_2}=\frac{\frac{1}{r_1}-\frac{1}{r}}{\frac{1}{r}-\frac{1}{r_2}}\cdot \qquad (4.13)$$

Преобразуем выражение (4.13), окончательно получим зависимость изменения температуры от радиуса удаления относительно нагретой точки:

$$T(r)=\frac{\left[T_1\cdot\left(\frac{1}{r_2}-\frac{1}{r}\right)-T_2\cdot\left(\frac{1}{r_1}-\frac{1}{r}\right)\right]}{\left(\frac{1}{r_2}-\frac{1}{r_1}\right)} \qquad (4.14)$$

Как видно из графика изменения температуры и формул для того чтобы процесс можно было считать близким к изотермическому необходимо максимально приравнять разницу тепературы газа и воды к одинаковому значению.

4.1.2 Моделирование ламинарного лечения сферического пузырька в цилиндрическом объёме вязкой жидкости

Изучением деформаций и течения материалов, включая эластичные, вязкие и пластичные свойства, занимается реология (18).

Вязкость – внутреннее трение. Это трение возникает между слоями жидкости при ее движении. Чем больше трение, тем больше силы необходимо приложить, чтобы вызвать движение («сдвиг»). Сдвиг имеет место при физическом перемещении или разрушении жидкости: разливе, растекании, разбрызгивании, перемешивании и т.п. Для сдвига жидкостей с высокой вязкостью необходимо приложить больше силы, чем для маловязких материалов.

Возникновение внутреннего трения в жидкостях на границах её движущихся друг относительно друга элементов обусловлено микрофизическим процессом передачи импульса от одних слоёв среды к другим. Но, кроме того, они участвуют и в беспорядочном тепловом движении. Количественно вязкость характеризует сопротивление жидкости перемешиванию её слоёв относительно друг друга. Вязкость является положительной величиной, и определяет скорость переноса импульса в результате теплового движения частиц жидкости. Вязкость численно равна импульсу, переносимому в единицу времени через площадку в 1 $м^2$ при градиенте скорости (в направлении, перпендикулярном к площадке), равном единице(1 м/с на 1 м длины).

В системе СИ единицей измерения коэффициента вязкости является Паскаль-секунда. Широко употребляется и единица измерения системы СГС. Она называется Пуазом (Пз) в честь французского учёного Ж. Пуазейля, впервые исследовавшего течение вязкой жидкости. Соотношение между единицами: 1 Па·с = 10 Пз.

Молекулярно-кинетическая теория объясняет вязкость движением и взаимодействием молекул (19). Молекулы жидкости, как и твердых тел, способны совершать колебания около положений равновесия. Колеблющаяся молекула жидкости может проникнуть в соседний слой, при этом необходимо затратить энергию W, которая называется энергией активации вязкого течения. Поскольку с изменением температуры частота и амплитуда колебаний молекул меняется, то вязкость будет величиной переменной. Вязкость большинства жидкостей с ростом температуры уменьшается по закону (20):

$$\eta = A \cdot \exp\left(\frac{W}{k \cdot T}\right), \qquad (4.15)$$

где k - постоянная Больцмана;

A - множитель, слабо зависящий от температуры T.

Вязкость на движение жидкости сказывается двояко: во-первых, она обеспечивает передачу движения от слоя к слою, благодаря чему скорость в потоке от точки к точке меняется непрерывно; во-вторых, переводит часть механической энергии потока в его внутреннюю энергию.

Внутренне трение является причиной того, что для протекания жидкости через трубу требуется создать разность давлений на её концах. Чтобы скорость течения имела некоторое значение, эта разность давлений должна быть тем больше, чем больше коэффициент внутреннего трения. Вязкость определяет быстроту передачи

импульса из одного слоя потока в другой. Скорость же равна импульсу, делённому на массу. Поэтому быстрота выравнивания скорости потока будет определяться величиной:

$$\nu = \frac{\eta}{\rho}, \qquad (4.16)$$

где ρ - плотность, то есть масса единицы объёма жидкости.

Величину ν называют кинематической вязкостью, в СИ она измеряется в м/с, а в системе СГС в Стоксах в честь английского физика Дж. Стокса.

Для описания физических свойств таких систем, необходимо определить дифференциальное уравнение движения вязкой жидкости. С этой целью проведем моделирование, которое обычно проводится при получении формулы Пуазейля. (18)

В качестве системы отсчета, возьмем сферический пузырек газа, который омывает однородный поток ламинарно-текущей жидкости. Для этого выделим цилиндрический объем вязкой среды некоторого радиуса *r* и длиной l (рисунок 4.5).

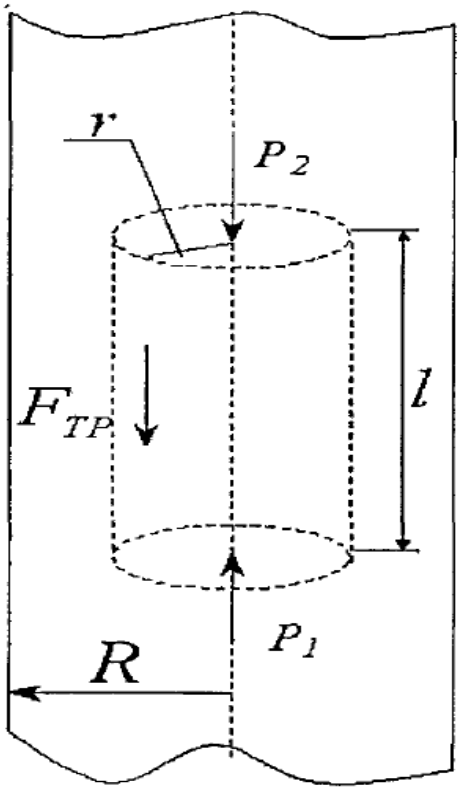


Рисунок 4.5 - Цилиндрический объем вязкой среды

Пузырек газа может находиться на торцах цилиндра, на которых поддерживается давление р 1 и р 2, что обусловливает действие сил давления, сумма которых равна:

$$F = S_1(p_1 - p_2), \qquad (4.17)$$

- где $S_1 = \pi \cdot r^2$ площадь основания цилиндра.

Тогда, перепишем (4.17), получим:

$$F = \pi \cdot r^2 \cdot (p_1 - p_2). \qquad (4.18)$$

На поверхность пузырька со стороны окружающего слоя вязкой среды действует сила внутреннего трения - сила Ньютона (18):

$$F_{TP} = \eta \cdot \frac{d\nu}{dr} \cdot S_2 = \eta \cdot \frac{d\nu}{dr} \cdot 2 \cdot \pi \cdot r \cdot l \qquad (4.19)$$

где η – динамическая вязкость

Для описания поведения газов при различных температурах и давлениях используется уравнение Ван-дер-Ваальса.

$$\left(p + \frac{a}{V_M^2}\right) = \frac{R \cdot T}{(V_M - b)}.$$

где p - давление действующее на пузырёк со стороны жидкости;

V_M - молярный объём газа в пузырьке;

T - температура газа впузырьке;

R - универсальная газоваяпостоянная;

a и b - константы Ван-дер-Ваальса, учитывающие отклонение свойств реального газа от свойств идеального.

Исходные данные для получения вычисления:

$\rho_ж$, кг/м^3	g, м/с^2	R, Дж/моль*К	T, К
9982	9,81	8,3145	293,15

давление над жидкой средой равно $p_{атм}$, либо $p_{атм} - p_1$ в случае использования двух компрессоров:

давление в моменте измерения равно: $p_{атм} - p_1 - \rho * g * h$

,где ρ – плотность жидкости,

g – ускорение свободного падения,

h- высота столба жидкости в моменте измерения

Так как объем сферы V = 4/3*π*R^3, получаем формулу для вычислениия объема одиночного пузырька:

$$V_г = \frac{4*\pi*(p_{атм} - p_1)*R^3}{3*(p_{атм} - p_1 - \rho*g*h - \frac{\sigma}{2}*R)} \quad (4.21)$$

Объемный расход газа можно представить как сумму объемов, вычисленных по формуле (4.21) за время проведения измерения.

$$Q = \frac{V_1 + V_2 + \cdots + V_n}{t} \quad (4.22)$$

Массовый расход:

$G = Q * \rho$, где ρ – плотность газа (4.23)

4.2. Экспериментальное обоснование дискретного метода измерения объема газа

4.2.1. Описание метода измерения и разработка средств измерения объемного расхода газа

Разрабатываемый метод и средство предназначены для измерения расхода и количества (объем) квазистацинарных потоков газов, у которых температура образования росы по воде не превышает температуру газа (для природного газа не выше чем по стандартам на газы), с вязкостью от $6x10^{-6}$ до 35×10^{-6} Пахе, а также реологических свойств жидкости, а именно – динамической вязкости и коэффициента поверхностного натяжения.

На основании результатов теоретического описания, сформированы ограничения математической модели, и как следствие самого метода контроля расхода газа путем барботирования. Требования к измерительной системе и проведению измерения:

форма пузырьков –сферическая;

пузырьки газа поднимаются вертикально вверх вдоль оси цилиндра;

каждый пузырек газа образуется в отдельности, отрывается и поднимается независимо от других, не образуя струю;

концентрация жидкости остается постоянной с течением времени;

точность измерения зависит от поверхностного натяжения, влияющего на процесс образования пузырька у границы сопла под действием температуры.

В соответствии с рекомендациями по порядку разработки методики выполнения измерений, а так же предложенной гипотезой (см. предыдущую главу) рассмотрим метод измерений, основанный на следующих принципах измерения и расчетных соотношениях.

4.3. Преобразование изображений

4.3.1. Оператор Собеля

Одна из наиболее базовых и важных сверток – это вычисление производных (или их приближенных значений). Существует множество способов сделать это, но только немногие из них хорошо подходят к конкретной ситуации. (21) (22) (23) (24)

В общем, наиболее частый оператор, используемый, чтобы получить дифференцирование – это производная Собеля (или оператор Собеля) (см. Рисунок 4-3 и

4-4). Оператор Собеля существует для производной любого порядка, а также и для смешанных частных производных (например $\partial^2/\partial x \partial y$).

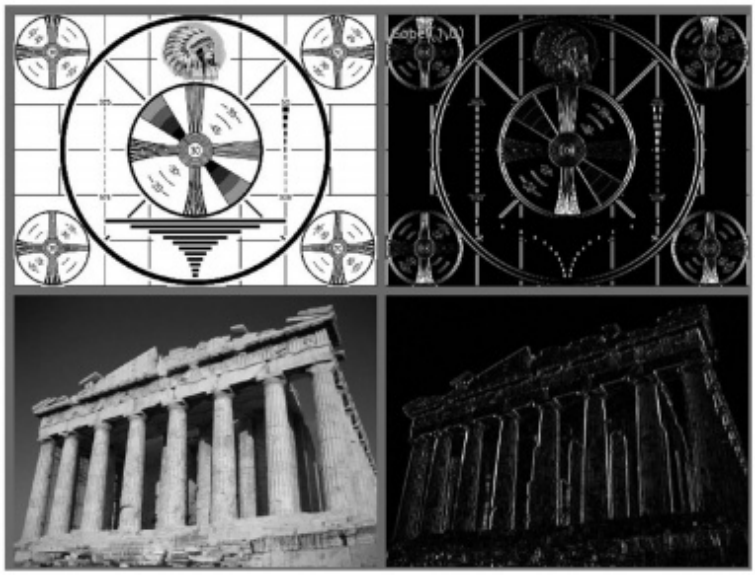


Рисунок 4-1. Результат применения оператора Собеля, используемый для аппроксимации производной по оси x.

int[][][] s*obelMapping* (int[][] grayImage);

Тут int[][] и int[][][] – ваши входное и выходное изображения соответственно.

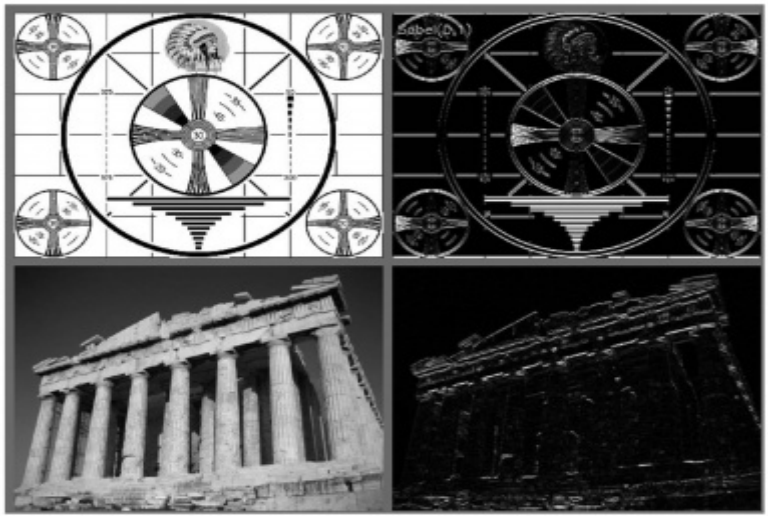


Рисунок 4-2. Результат применения оператора Собеля, используемый для аппроксимации производной по оси y.

Производные Собеля имеют хорошее свойство в том смысле, что они могут быть определены для ядер произвольного размера и эти ядра могут быть построены быстро и итерационно. Бóльшие ядра дают лучшую аппроксимацию производной, т.к. меньшие ядра намного более чувствительны к шумам изображения.

Чтобы понять это более точно, мы должны представить, что производная Собеля в действительности производной не является. Все потому, что оператор Собеля определен

на дискретном пространстве. Что в действительности представляет собой оператор Собеля - это подгонку под полином. Так что, производная Собеля второго порядка, - это не вторая производная, а локальная подгонка под параболическую функцию. Это объясняет почему может понадобиться использовать ядро большего размера, т.к. большее ядро производит подгонку над бóльшим количеством пикселей.

4.3.2. Фильтр Щарра

В действительности существует множество способов аппроксимировать производную для случая дискретной сетки. Недостатком аппроксимации используемой в операторе Собеля является то, что она менее точная для маленьких ядер. Для больших ядер, где используется больше точек для аппроксимации, эта проблема менее значительна. Эти неточности не вырисовываются для фильтров по осям *X* или *Y*, используемых в s*obelMapping()*, поскольку они точно выровнены по осям *x* и *y*. Осложнения появляются, когда вы хотите провести замеры по изображению с использованием аппроксимации производной по направлению (например, направление градиента изображения путем использования результата фильтра по арктангенсу *y/x*).

Чтобы прояснить смысл, реальным примером где вам могут понадобиться замеры изображений такого вида – это процесс получения информации о форме объекта путем сбора гистограммы градиентных углов вокруг объекта. Подобные гистограммы – это входные данные и основа над которой свою работу выполняют множество обычных классификаторов форм (как для их обучения, так и для классификации). При таком применении, неточные измерения угла градиента будут снижать качество (и эффективность) классификации.

Для фильтра Собеля размером 3×3 неточности растут с увеличением градиентного угла (по направлению от горизонтального к вертикальному). Фильтр Щарра настолько же быстр, но более точен чем фильтр Собеля, так что более предпочтительно его использовать, если проводить подобные измерения по изображению используя фильтр размером 3×3. Коэффициенты Щарра показаны на Рисунке 4-5.

-3	0	3
-10	0	10
-3	0	3

-3	-10	-3
0	0	0
3	10	3

Рисунок 4-5. Фильтр Щарра размером 3×3.

Код для заполнения матрицы 3x3 для фильтра Собеля:

```
public int[][][] sobelMapping(int[][] grayImage)
{
   int[][][] result = new int[2][grayImage.length][grayImage[0].length];
   for(int j = 0; j<grayImage.length; j ++)
   {
      for(int i = 0; i<grayImage[0].length; i ++)
      {
         if( !(j - 1 < 0 || j+1 >= grayImage.length
               || i - 1 < 0 || i + 1 >= grayImage[0].length))
         {
            result[0][j][i] = (grayImage[j-1][i-1] *   sobelX[0][0] +
                  grayImage[j-1][i]   *   sobelX[0][1] +
                  grayImage[j-1][i+1] *   sobelX[0][2] +
                  grayImage[j][i-1]   *   sobelX[1][0] +
                  grayImage[j][i]     *   sobelX[1][1] +
                  grayImage[j][i+1]   *   sobelX[1][2] +
                  grayImage[j+1][i-1] *   sobelX[2][0] +
                  grayImage[j+1][i]   *   sobelX[2][1] +
                  grayImage[j+1][i+1] *   sobelX[2][2]);

            result[1][j][i] = (grayImage[j-1][i-1] *   sobelY[0][0] +
                  grayImage[j-1][i]   *   sobelY[0][1] +
                  grayImage[j-1][i+1] *   sobelY[0][2] +
                  grayImage[j][i-1]   *   sobelY[1][0] +
                  grayImage[j][i]     *   sobelY[1][1] +
                  grayImage[j][i+1]   *   sobelY[1][2] +
                  grayImage[j+1][i-1] *   sobelY[2][0] +
                  grayImage[j+1][i]   *   sobelY[2][1] +
                  grayImage[j+1][i+1] *   sobelY[2][2]);
         }
         else
         {
            result[0][j][i] = 0;
            result[1][j][i] = 0;
         }
      }
   }
   return result;
}
```

4.3.3. Преобразование Лапласа

Функция Laplacian (впервые использованную в компьютерном зрении Марром – представляет собой дискретную реализацию оператора Лапласа, аналог непрерывного оператора Лапласа:

$$\text{Laplace}(f) \equiv \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Поскольку оператор Лапласа может быть определен в терминах вторых производных, дискретная реализация работает подобно производной Собеля второго порядка. В действительности так и есть, и реализация оператора Лапласа в использует оператор Собеля в своих расчетах.

Функция принимает в качестве входных параметров изображение-источник и изображение-приемник, а также размер апертуры. Исходное изображение может быть либо 8-битным (беззнаковым), либо 32-битным (вещественным *float*) изображением. Выходное изображение должно быть 16-битным (знаковым) либо 32-битным (вещественным) изображением. Параметр *appertureSize* – это точно та же апертура, которая появлялась в производных Собеля, и в результате задает размер региона по которому пиксели обрабатываются при вычислении вторых производных.

Оператор Лапласа может быть использован во множестве приложений. Общее применение – это обнаружение «пузырей». Вспомните, что оператор Лапласа выражается в виде суммы вторых производных по осям *x* и *y*. Это означает, что единичная точка или любой маленький пузырек (меньший апертуры), который окружен бóльшими значениями приведет к возрастанию значения выходной функции. И наоборот, точка или маленький пузырек, который окружен меньшими значениями будет стремиться привести значение выходной функции к отрицательным значениям.

Принимая это во внимание, оператор Лапласа может быть использован в качестве детектора для подсветки (выделения) границ. Чтобы увидеть, как это происходит, представьте себе первую производную функции, которая будет больше в тех местах, где функция быстро изменяется. В то же время, она будет скачкообразно возрастать в тех местах где мы приближаемся к краеобразной неоднородности, и уменьшаться скачкообразно, как только мы выходим из неоднородности. Выходит, что производная будет получать локальный максимум где-то в этом диапазоне. Таким образом, мы можем искать нули вторых производных для того чтобы найти такие локальные максимумы. Грани в исходном изображении будут нулями для Лапласиана. К сожалению, более существенные и менее значимые будут нулями для Лапласиана, но это не проблема, т.к.

мы можем просто отфильтровать те пиксели, которые имеют бóльшие значения для первой (Собелевой) производной. На Рисунке 4-6 приведен пример применения Лапласиана над изображением вместе с деталями о первых и вторых производных и их пересечения нулевых значений.

4.3.4. Алгоритм Канни

Только что описанный метод поиска границ был в дальнейшем улучшен Дж. Канни в 1986 году и теперь называется "детектор границ Канни" (24) (21). Одно из отличий алгоритма Канни от более простого, основывающегося на преобразовании Лапласа в том, что в алгоритме Канни первые производные вычисляются по осям *x* и *y* а затем комбинируются в четыре производных по направлению. Точки, в которых эти производные достигают локального максимума затем рассматриваются в качестве кандидатов на группировку в грань.

Однако наиболее значимым дополнением алгоритма Канни является то, что он пытается собрать кандидаты на грани независимых пикселей в целые контуры. Эти контуры формируются путем применения к пикселям пороговой функции с гистерезисом. То есть существуют два порога, - верхний и нижний. Если градиент пикселя имеет значение большее, чем верхний порог, значит он включается в грань. Если градиент ниже чем нижний порог, то данный пиксель отбрасывается. Если градиент пикселя находится между порогами, то он будет рассматриваться как грань только если он соединен с пикселем, который лежит выше верхнего порога. Согласно рекомендации автора (Канни), отношение порогов *высокий:низкий* должно быть между 2:1 и 3:1. На рисунках 4-7 и 4-8 показаны результаты применения process_fast() к тестовому узору и фотографии (с соотношениями порога гистерезиса *высокий:низкий* равными 5:1 и 3:2 соответственно).

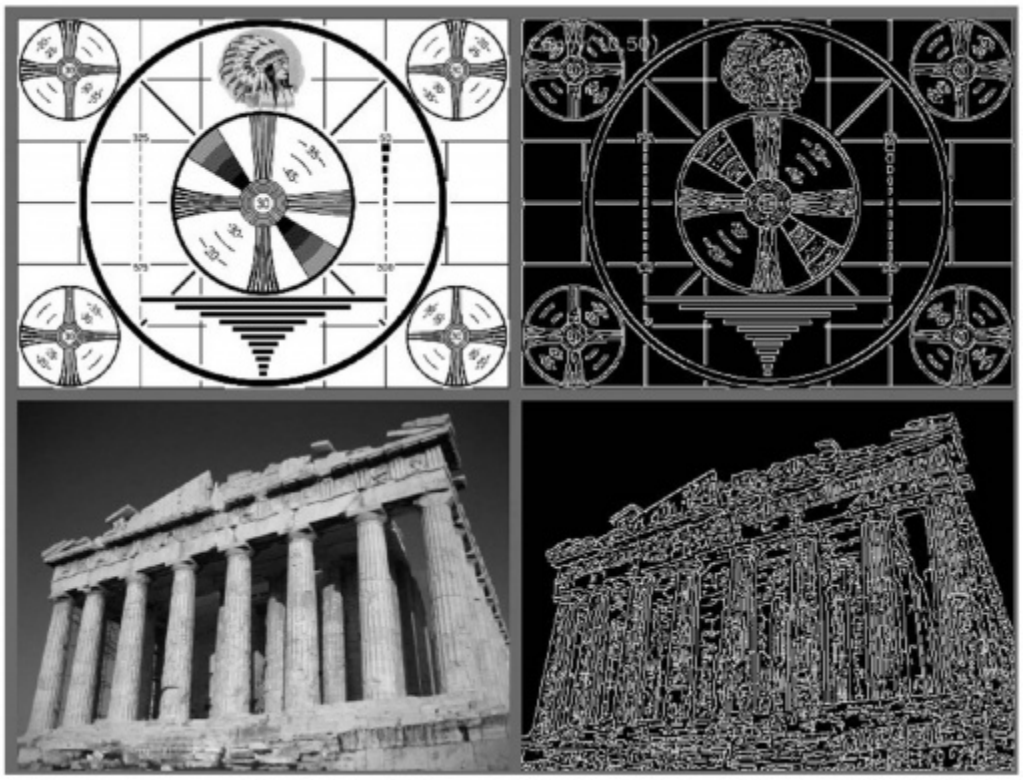


Рисунок 4-6. Результат применения детектора граней Канни для двух разных изображений, при котором верхний и нижний пороги установлены равными 50 и 10 соответственно.

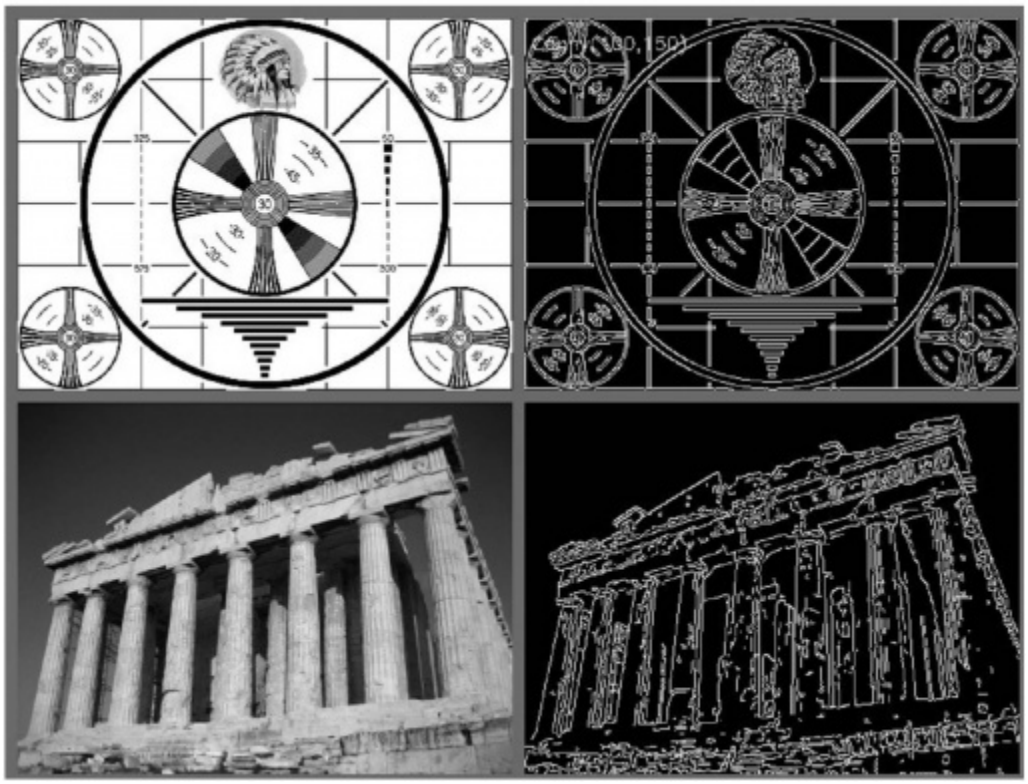


Рисунок 4-7. Результат применения детектора граней Канни для двух разных изображений, при котором верхний и нижний пороги установлены равными 150 и 100 соответственно.

Функция *process_fast(int[] data_in, int w, int h) класса CED_fast* принимает входное изображение в оттенках серого. Следующие два параметра – это высота и ширина изображения. Отметим однако, что *process_fast()* в действительности не возвращает линии контура, - если они нам понадобятся, мы должны будем их получить сами путем вызова *g.drawImage(cannyImage, 0, img.getHeight(height);*

```
public void process_fast(int[] data_in, int w, int h) {
      width = w;
      height = h;
      picsize = width * height;
      initArrays();
      readLuminance_fast(data_in);
      if (contrastNormalized) normalizeContrast();
      computeGradients(gaussianKernelRadius, gaussianKernelWidth);
      int low = Math.round(lowThreshold * MAGNITUDE_SCALE);
      int high = Math.round(highThreshold * MAGNITUDE_SCALE);
      performHysteresis(low, high);
      thresholdEdges();
      writeEdges(data);
   }
```

4.3.5. Преобразование Хафа

Преобразование Хафа для круга (окружности) (24) (22) (23) работает почти аналогично преобразованию Хафа для линий. Плоскость накопителя будет замещена плоскостью объема с тремя измерениями: одно для *x*, одно для *y*, и последнее для радиуса круга *r*. Это означало бы значительно большие запросы памяти и значительно меньшую скорость. Реализация кругового преобразования это хитрый метод, называемый *градиентным методом Хафа*.

Градиентный метод Хафа работает следующим образом. Вначале изображение проходит фазу поиска граней (в нашем случае, *process_fast()*). Затем, для каждой ненулевой точки изображения граней ищется локальный градиент. (Градиент вычисляется путем расчёта производных Собеля первого порядка для *x* и *y*). Используя градиент, каждая точка линии выраженная по этому наклону (от определенного минимума расстояний до определенного максимума расстояний) увеличивает значение в плоскости накопителя на единицу. В то же время, запоминается расположение каждой ненулевой точки на изображении граней. Центры-кандидаты затем выбираются из тех точек (двухмерного) накопителя, величины которых выше заданного порога и, одновременно, больше всех их непосредственных соседей. Эти центры-кандидаты размещаются в порядке убывания их величин накопителя, так что центры, которые имеют наибольшее количество поддерживающих пикселей, появляются первыми. Далее, для каждого центра, рассматриваются все ненулевые пиксели (вспомните, что их список был построен ранее). Эти пиксели упорядочиваются в соответствии с их расстоянием к центру. Проходя от наименьших расстояний до наибольших радиусов, выбирается единственный радиус, который лучше всего поддерживается ненулевыми пикселями. Центр сохраняется, если он имеет достаточную поддержку ненулевых пикселей на изображении граней, и если он имеет достаточное расстояние от любого ранее найденного центра.

Эта реализация позволяет алгоритму выполняться намного быстрее и, возможно более важно, позволяет обойти проблему разросшегося трехмерного накопителя, который бы привел к значительно большему шуму и нестабильно отражал бы результат. С другой стороны, этот алгоритм также имеет свои недостатки, поэтому реализованный в данной работе алгоритм Канни имеет достаточную точность выявления сферических объектов в представленном изображении при высокой скорости работы.

4.3. Вывод

Очевидно необходимо процесс максимально приблизить к изотермическому чтобы снизить влияние температуры к минимуму.

Полученные графические зависимости изменения радиуса пузырька и его подъёмной силы на различных глубинах позволяют сделать следующие выводы: диаметр пузырька изменяется при подъёме с глубины, рост подъёмной силы происходит быстрее в сравнении с ростом объёма.

Проведенный анализ теории ламинарного течения жидкости и капиллярных явлений на границе раздела фаз позволил выявить комплекс параметров и определить ряд зависимостей, позволяющих проводить измерение расхода газа с учётом влияния реологических свойств жидкости на объём газового пузырька, перемещающегося в среде.

5. Экологическая часть

В экологической части рассмотрены влияние на человека и окружающую среду, побудителя расхода МКМ-7. Показания величин вредных факторов сравниваются с допустимым и уровнями. Предложены варианты нейтрализации вредных факторов, превышающих допустимые уровни.

5.1. Анализ шумового воздействия МКМ-7

В процессе работы МКМ-7 издаёт тональный шум. Шум вызван колебательным движением мембраны и сердечника катушки. В процессе работы уровень звукового давления шума был определён экспериментально. Уровень громкости определялся с помощью приложения для мобильных устройств «Шумомер» Green Liston. Так как звук вызван колебанием мембраны, то его частота совпадает с частотой колебательного движения мембраны и, соответственно, коррелируют с частотой сигнала управляющего напряжения на катушке МКМ-7. В результате опыта совпадение частот было подтверждено экспериментально.

Результаты измерения для различных режимов питания МКМ-7 при различных частотах звука представлены на рисунке 39. На график нанесена кривая предельно допустимого уровня звукового давления при различных частотах звука для деятельности с высокой умственной нагрузкой, указанной в СН 2.2.4/2.1.8.562-96. Эта кривая соответствует наиболее строгому требованию из указанных в СН. Как видно из полученного графика, шум побудителя расхода ни на одном из исследованных режимов не превышает допустимого уровня.

В ходе работы проводилось исследование плотности спектральной мощности шума. На рисунке 40 представлены входные и выходные данные для программы определения спектральной мощности шума. В качестве входного параметра используется звуковой файл в формате wav. Это шум МКМ-7, записанный с помощью микрофона.

Наиболее информативными оказались измерения, проведённые при шуме с частотами 50, 250 и 430 Гц. Они показаны на рисунке 41.

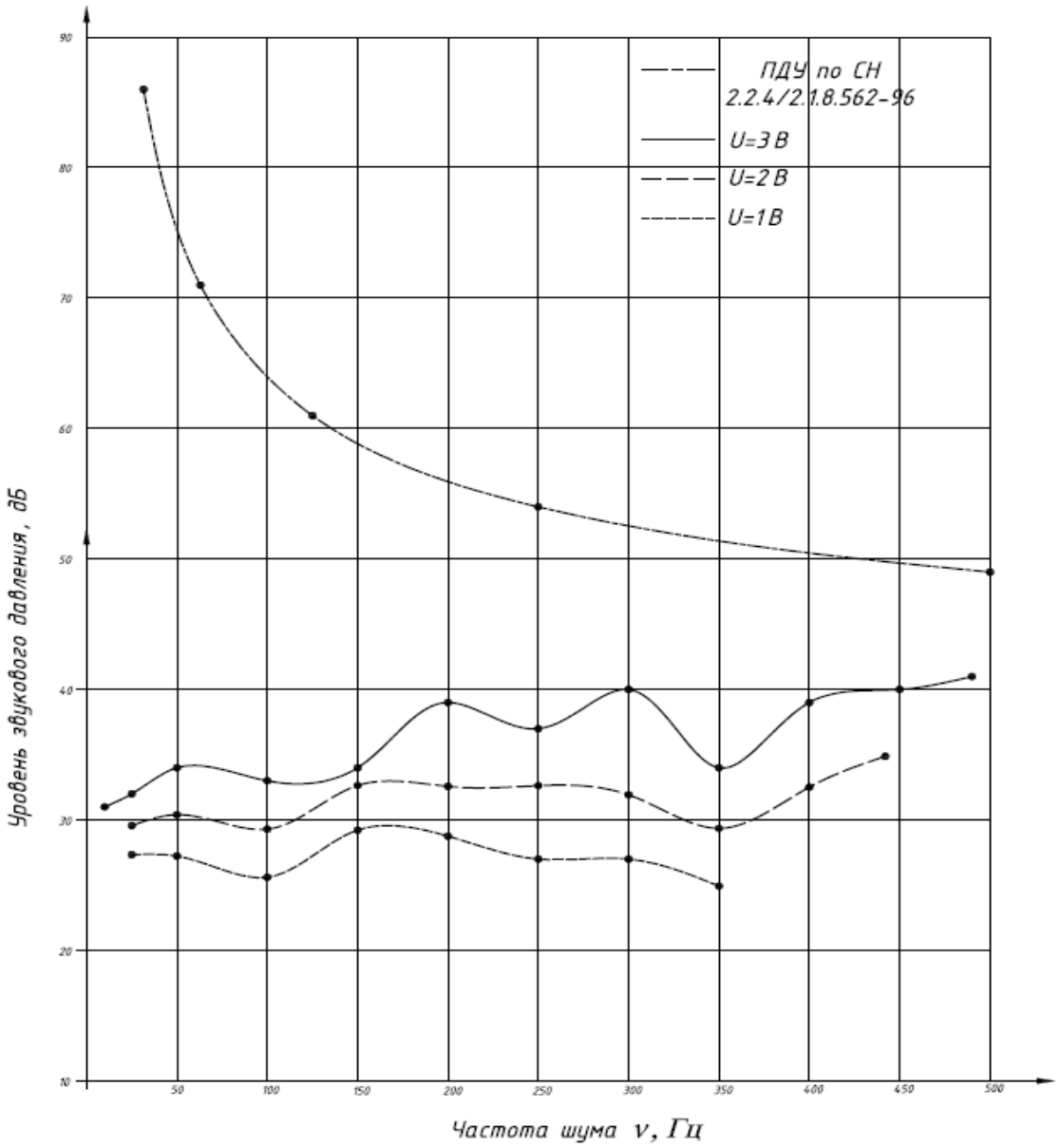


Рисунок 39. Зависимость уровня звукового давления от частоты звука.

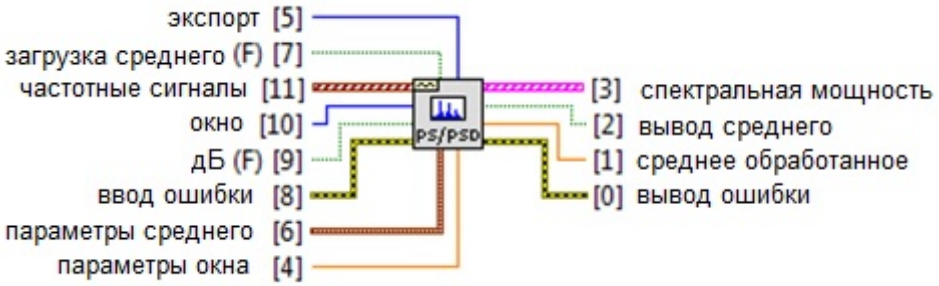


Рисунок 40. Определение спектральной мощности шума.

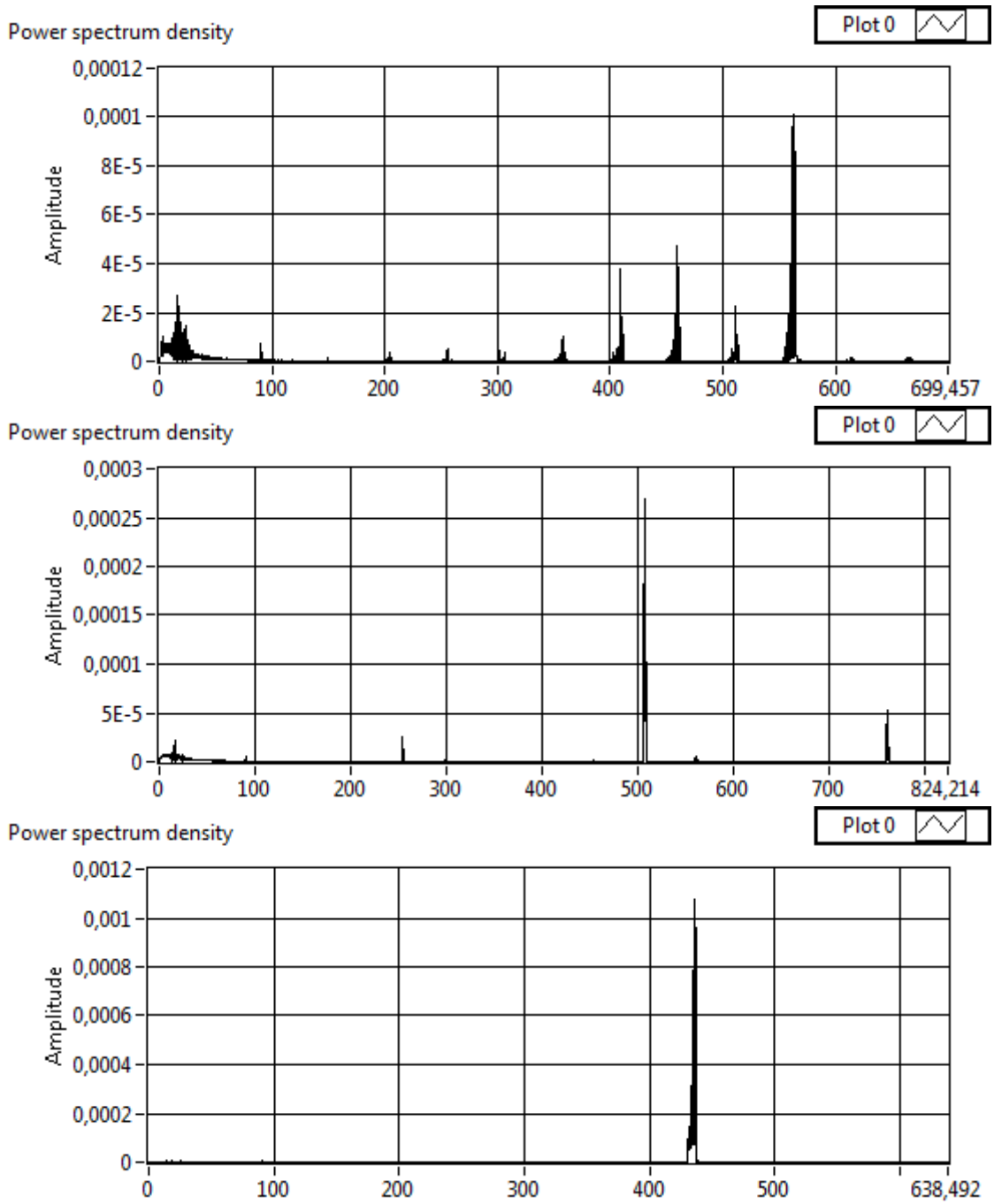


Рисунок 41. Плотность спектральной мощности звука при частотах 50, 250 и 430 Гц (сверху вниз).

Результаты показаны на **графическом листе 6**. На графике для управляющего сигнала частотой 50 Гц видны скачки плотности в районе частоты 50 Гц, 100 Гц, 150 Гц и далее, то есть в районе частоты, кратной 50. Это может быть связано с отскоком мембраны от ограничительного диска.

На графике для частоты 250 Гц так же видны подобные скачки при 500 Гц и 750 Гц, что кратно частоте колебаний мембраны.

На всех трёх графиках виден значительный скачок плотности в районе частоты 430-560 Гц. Это объясняется наличием собственных колебаний колебательной системы пружина-сердечник-мембрана. При этих частотах мы наблюдаем явление резонанса, обусловленного совпадением частоты управляющего напряжения на катушку и собственными колебаниями система пружина-шток. Подтверждение эффекта резонанса можно наблюдать в подпункте 4.2**.** Наиболее явно эффект наблюдается на графике зависимости перепада давления от частоты управляющего сигнала в режиме вакуумного насоса (рисунок 42).

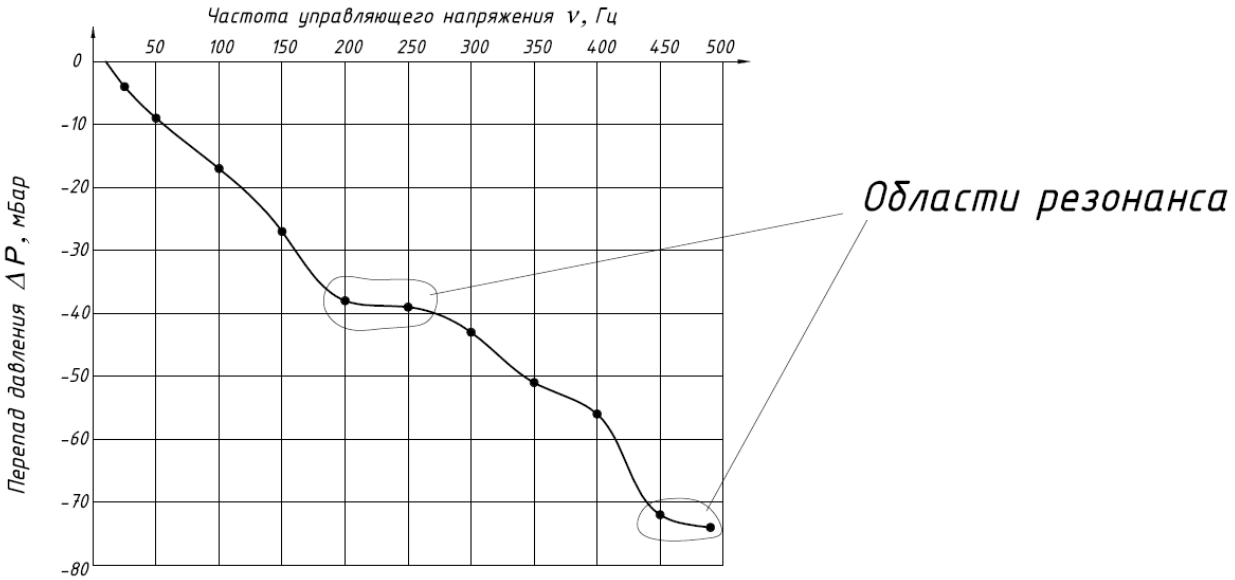


Рисунок 42. Влияние резонанса на характеристики МКМ-7.

В районе частоты 200-250 Гц и 450-490 Гц перепад давлений остаётся почти неизменным, что может быть связано с описанным выше резонансом, который вызывает рассогласование движения мембраны и сигнала управляющего напряжения. Это подтверждает эффект, описанный в подпункте 4.2.

5.2. Анализ прочих экологических факторов.

МКМ-7 – мембранный компрессор, одной из главных особенностей которого является полное отсутствие масла в рабочей полости компрессора, (25). Это значит, что газ, над которым совершает работу микрокомпрессор, не загрязняется парами углеводородов. К тому же это означает, что МКМ-7 не изменяет качественный состав перекачиваемого газа, то есть не влияет на точность показаний сенсора побудителя расхода, а также может применяться в иных системах, где недопустимы изменения в составе перекачиваемого газа.

Вибрационное воздействие на оператора и прочие элементы устройства пренебрежимо мало.

В процессе работы МКМ-7 выделяет тепло. Её источником является управляющая катушка. По данным производителя (26), к поломке устройства нагрев не приводит. Однако при величине управляющего напряжения катушки более 15 В МКМ-7 может нагреваться до температуры более 60℃ (25) Длительный контакт с поверхностью с такой температурой способен вызывать ожоги первой степени, а также повреждение прочих элементов устройства. Поэтому при использовании МКМ-7 производителем накладывается ограничение по величине управляющего напряжения на катушке – не более 12 В. В ходе работы МКМ-7 не испытывался при режимах питания с напряжением более 3 В, поэтому фактором нагрева устройства можно пренебречь.

5.3. Вывод.

Основной негативный фактор, воздействующий на оператора и окружающую среду это шум побудителя расхода (микрокомпрессор МКМ-7). Определены уровни звукового давления шума при различных режимах работы МКМ-7. Каких-либо режимов работы, при которых МКМ-7 издаёт превышающий допустимые уровни шум, не выявлено. Однако уровень воздействия МКМ-7 не превышает предельных уровней, определенных СН 2.2.4/2.1.8.562-96. В самой измерительной системе не выявлено каких-либо негативных факторов влияющих на экологию и окружающую среду.

6. Экономическая часть

6.1. Оценка стоимости измерительной системы расхода газа

В общем случае измерительная система может быть собрана из следующих компонентов (Цены в рублях на 01.08.2015):

- измерительная кювета (состав: стекло) – 150 руб.
- герметик для стекол – 400 руб.
- побудитель расхода (в стоимость измерительной системы не включается)
- видеокамера (30 fps) – 900 руб.
- ПО для распознавания и отслеживания пузырьков газа - 0 руб.

Итого: 1450 руб.

6.2. Экономическая мотивация использования предлагаемой измерительной системы расхода газа

Существующие измерительные системы расхода аналогичные той, которая представлена в данной работе и имеющие аналогичную точность отличаются большой стоимостью. Так, например, запатентовано изобретение для измерения расхода газов с оптическим датчиком расхода в виде измерительной кюветы с жидкостью, трубопровода с соплом, источником света и фотоприемником. Однако такая конструкция кюветы более сложная, а также входящая в измерительную систему схема счета газовых пузырьков. См. рис 6.1. (27)

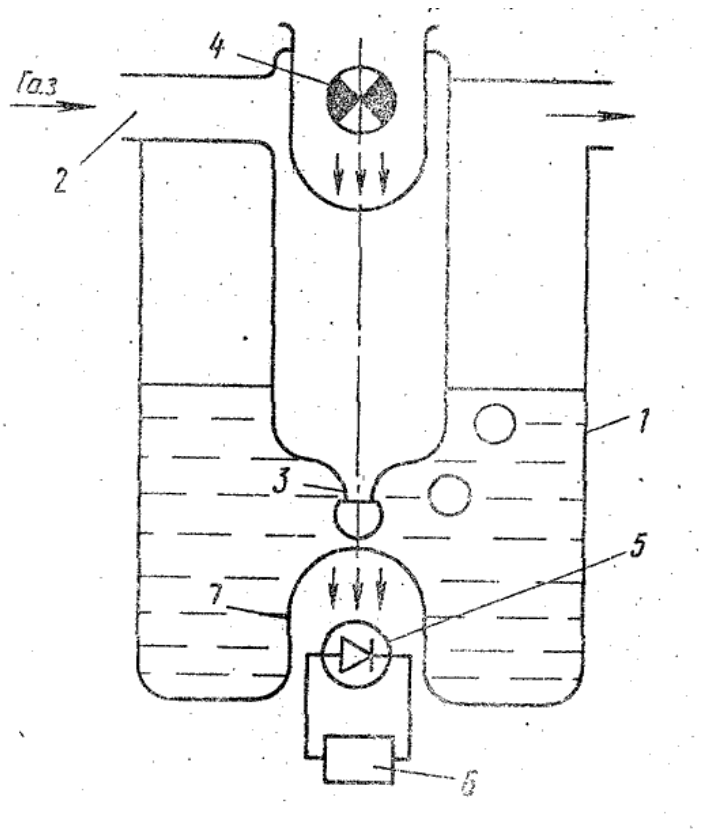


Рис. 6.1

Еще один аналогичный датчик расхода содержит дополнительный фотоприемник и фотодиод и устройство счета см. рис 6.2 (28)

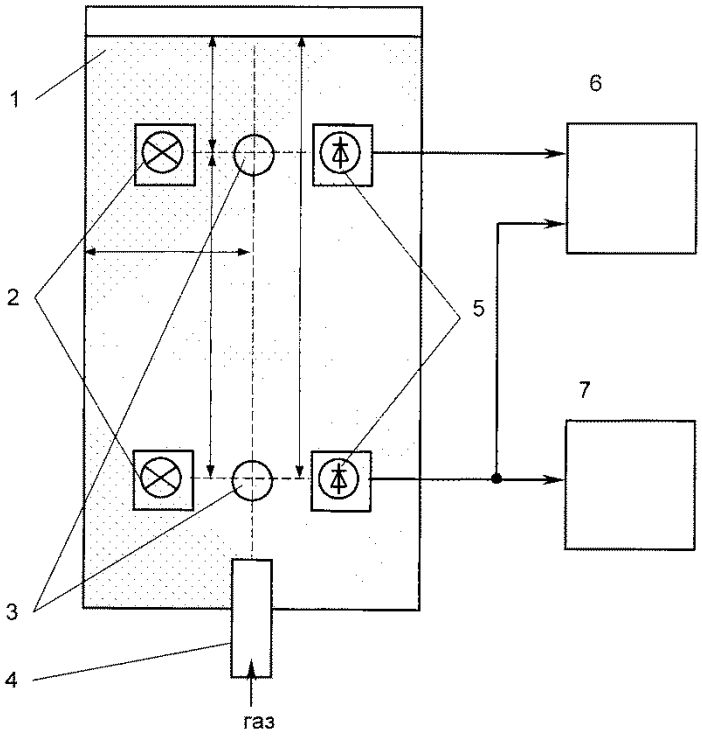


Рис. 6.2

Все вышеприведенные датчики имеют большую стоимость в отличие от предлагаемого в данной работе.

6.3. Вывод

Данный оптический датчик расхода обладает преимуществом по сравнению с существующими аналогами, имеет более простую конструкцию при аналогичной точности получаемых результатов измерений.

7. Заключение

В заключении представлены полученные результаты исследования:

1. Обзор средств измерения расхода газа в малых и сверхмалых объемах показал, что пузырьковый метод обладает преимуществами по сравнению с существующими волюмометрическим и манометрическим методами. К основным преимуществам следует отнести высокую точность измерения, возможность регистрации кривой с самого начала процесса газовыделения.
2. В конструкторской части представлена конструкция измерительной системы расхода газа, описаны требования к конструкции и основные компоненты. Схематично представлены этапы выделения сферических объектов с изображения. Представлены пути программного решения данной алгоритмической задачи.
3. В технологической части работы был разработан испытательный стенд для определения объемного расхода газа, разработана методика испытания измерительной системы и методика обработки данных, получаемых при испытании.
4. В исследовательской части работы был проведен анализ теории ламинарного течения жидкости и капиллярных явлений на границе раздела фаз, который позволил выявить комплекс параметров и определить ряд зависимостей, позволяющих проводить измерение расхода газа с учётом влияния реологических свойств жидкости на объём газового пузырька, перемещающегося в среде, проведено исследование процессов теплообмена в системе, состоящеё из жидкой и газообразной фаз. Разработано устройство измерения малых и сверхмалых расходов газа.
5. В экологической части рассмотрено влияние МКМ-7 на человека и окружающую среду, сделан вывод об экологической безвредности устройства при любых режимах работы.
6. В экономической части сделан вывод о целесообразности применения измерительной системы для измерения расхода газа в малых и сверхмалых объемах.

Список литературы

1. **Л.Д., Ландау.** *Теоретическая физика: Механика сплошных сред. 2-е изд. перераб. и доп.* Москва : ГосТехИздат, 1953.

2. **Кюрджиев Ю.В., Чернышёв А.В.** *Исследование процесса истечения газа из ёмкости постоянного объёма через дроссельную шайбу в окружающую среду. Методические указания для выполнения лабораторной работы по курсу «Теория и расчёт процессов в агрегатах пневматических систем».* Москва : б.н., 2013.

3. **А.Ш., Киясбейли.** *Вихревые счетчики и расходомеры.* Москва : Машиностроение , 1974.

4. **П.А., Коротков.** *Тепловые расходомеры.* Ленинград : Машиностроение, 1969.

5. **П.П., Кремлевский.** *Расходомеры и счетчики количества.* Ленинград : Машиностроение, 1989.

6. **Г.С., Бондарев.** *Дискретный метод измерения малых объемов газа.* б.м. : Измерительная техника, 1970.

7. **Бастль, В.** *Измерения в промышленности. В 3 кн. Кн 2: способы измерения и аппаратура: Справочник.* б.м. : Металлургия, 1990.

8. **2939-63, ГОСТ.** Газы. Условия для определения объема. 1988 : Издательство стандартов, Москва r.

9. **15528-86, ГОСТ.** *ГОСТ 15528-86 Средства измерения расхода, объема или массы, протекающих жидкостей или газа. Термины и определения.* б.м. : Издательство стандартов, 1987.

10. Honeywell Inc. Pressure sensors 40PC series. Freeport, Illinois.

11. **Дж., Тэйлор.** *Введение в теорию ошибок.* Москва : Мир, 1985.

12. **Зайдель.** *погрешности измерений случайных величин.* Ленинград : Наука, 1985.

13. **А., Померанцев.** *Статистика.* Москва : Российское хеммометрическое общество, 2011.

14. **В.П., Исаченко.** *Теплопередача.* Москва : Энергия, 1975.

15. **В.В., Степанов.** *Курс дифференциальных уравнений.* Москва : Физ-мат лит., 1958.

16. **Лаптев Г.И., Лаптев Г.Г.** *Уравнения математической физики.* Москва : С, 2003.

17. **Г.Н., Ватсон.** *Теория бесселевских функций.* Москва : Иностранная литература. пер. с англ., 1949.

18. **И.В., Савельев.** *Курс общей физики Т1. механика, колебания и волны, молекулярная физика.* Москва : Наука, 1970.

19. —. *Курс общей физики. Учебное пособие в 3 томах. Том 1. механика, молекулярная физика.* Москва : Наука, 1987.

20. **Н.Б., Варгафтик.** *Справочник по теплофизическим свойствам газов и жидкостей.* Москва : Физматгиз, 1963.

21. **Gary Bradsky, Adrian Kaehler.** *Learning OpenCV: Computer Vision with the OpenCV Library 1 изд.* б.м. : O`Reilly.

22. **Brahmbhatt, Samarth.** *Practical OpenCV (Technology in Action) 1 изд.*

23. **Laganiere, Robert.** *OpenCV Computer Vision Application Programming Cookbook, 2-е изд.*

24. **Dawson-Howe, Kenneth.** *A Practical Introduction to Computer Vision with OpenCV (Wiley-IS&T Series in Imaging Science and Technology).* б.м. : Wiley.

25. **Алтухов С.М., Румянцев В.А.** *Мембранные компрессоры.* Москва : Машиностроение, 1967.

26. **С.М., Алтухов.** *Мембранные компрессоры.* Москва : Машиностроение, 1957.

27. **В.А., Андреев.** *Оптический датчик расхода. 1742624* Россия, 23 09 1992 г.

28. **С.Ф., Корндорф.** *Устройство измерения расхода газа. 2366902* Россия, 13 02 2008 г.

29. **Насибуллин С.Р., Чернышёв А.В.** арпель 2010 г., Вакуумная, компрессорная техника и пневмоагрегаты. Сборник трудов III Всероссийской молодёжной научно-практической конференции.

30. **О.Д., Балдин А.А.Бражников Н.И.Крылова.** *Вопросы конструирования акустических преобразователей расхода. техн. конференция.* Москва : б.н., 1973.

Приложение А. Руководство по установке необходимых программных компонентов

Программа написана на зыке Java, который является платформо - независимым языком, однако требует предустановленной программной среды Java Development Kit (JDK). Подойдет любая версия JDK начиная с JDK 1.6 и выше. JDK является бесплатным продуктом, распространяется компанией Oracle и доступен для скачивания по ссылке: http://www.oracle.com/technetwork/java/javase/downloads/index.html

Необходимо выбрать 32-х разрядную версию.

Установка достаточно простая. Инструкция по установке прилагается на странице загрузки.

После установки необходимо добавить переменную окружения JAVA_HOME: Система->Дополнительные параметры системы-> переменные среды ->Системные переменные: Имя: JAVA_HOME Значение: путь к каталогу java: например: C:\Program Files (x86)\JDK1.x.x для ОС Windows.

После установки jdk необходимо установить расширение Jmf (java media framework), который доступен для скачивания по ссылке: http://www.oracle.com/technetwork/java/download-142937.html

Далее необходимо также добавить переменную окружения JMF_HOME: Система->Дополнительные параметры системы-> переменные среды ->Системные переменные: Имя: JMF_HOME Значение: путь к каталогу java: например: C:\Program Files (x86)\JMFx.x.x для ОС Windows.

Настройка видеокамеры осуществляется с помощью приложения jmfregistry.exe, которое находится в папке bin каталога JMFx.x.x.

На вкладке Capture Devices необходимо нажать кнопку Detect Capture Devices. Программа выполнит поиск подходящих устройств и добавит их в реестр JMF.

Запуск программы осуществляется запуском исполняемого файла, например bat - файла для ОС Windows.

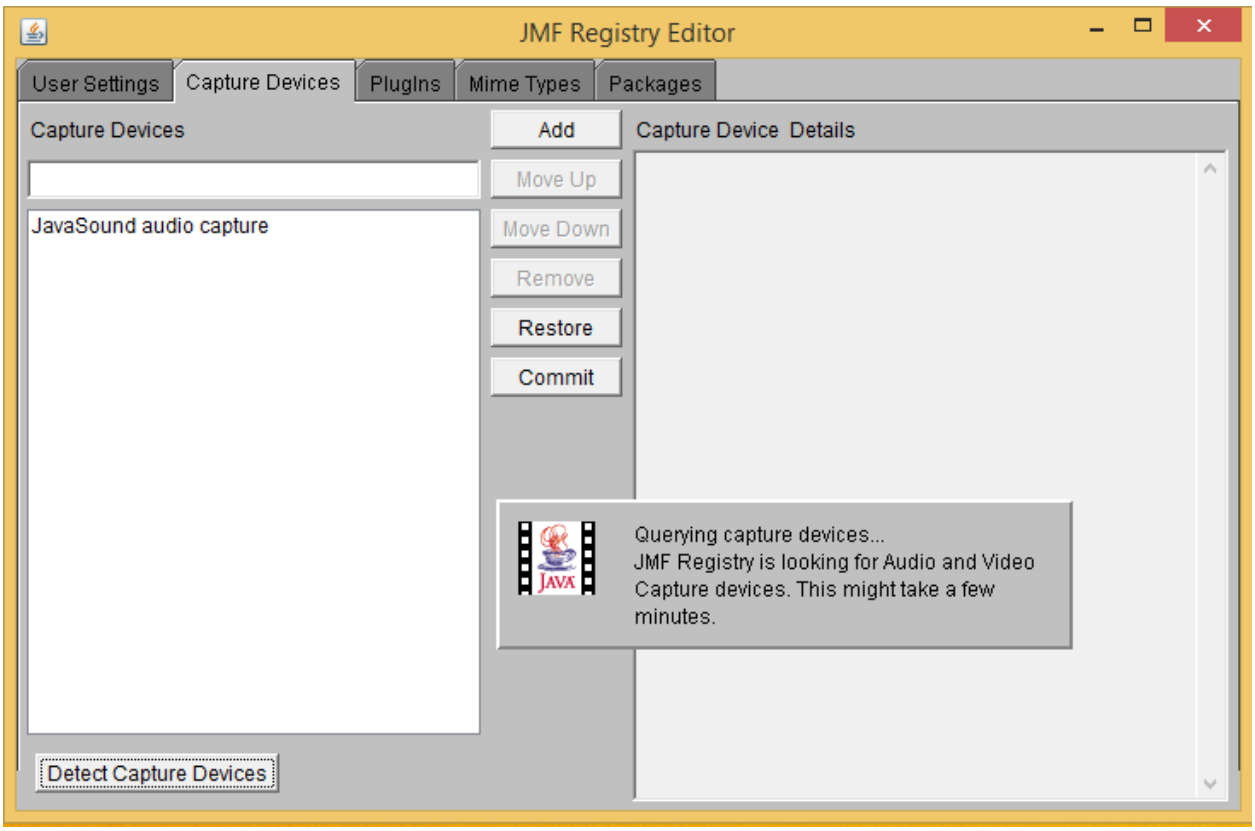


Рис А1. Настройка видеокамеры в Java Media Framework

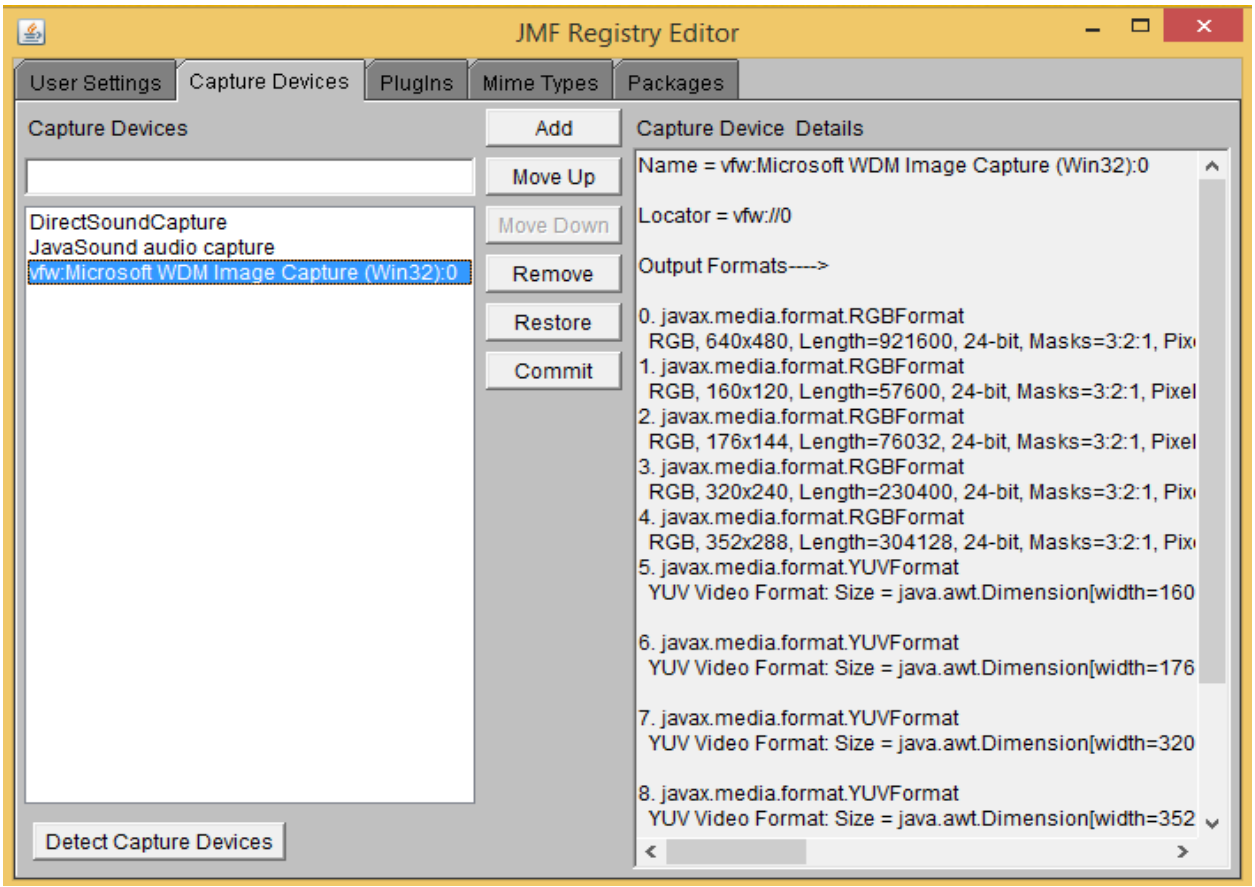


Рис А2. Настройка видеокамеры в Java Media Framework

Приложение Б. Код программы, управляющий вычислением расхода газа

Класс BubbleCatcherMain.java

```
/*
* Sergey Lavrinov 15-10-2015
* */

import javax.media.*;
import javax.media.format.*;
import java.util.Enumeration;
import javax.media.control.*;
import javax.media.util.*;

import java.awt.image.*;
import java.util.Vector;
import java.awt.*;
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class BubbleCatcherMain {

   CaptureDeviceInfo device;
   MediaLocator ml;
   Player player;
   FrameGrabbingControl fgc;
   JPanel pPanel;

   public int currFPS = 0, lastFPS = 0;
   public int cameraPosition = 75;
   public int maxCameraPosition = 145;
   public boolean trackBubble = false;

   public boolean isCapturingVideo = false;
   public int serialMoveMethod = 0;
   public static final int RADIUS = 0, XSPOT = 1;

   public Font timesNewRomanFont12 = new Font("Times New Roman", Font.PLAIN, 12);
   public Font timesNewRomanFont14b = new Font("Times New Roman", Font.BOLD, 14);

   public static void main(String args[]) {
      getList();
      new BubbleCatcherMain();
   }

   BubbleCatcherMain() {

      try {

         Frame window = new Frame();
         window.setBounds(10, 10, 800, 600);
         window.setTitle("Бакалаврская работа: Измерение объемного расхода газа в малых и сверхмалых объемах");
         window.setLocation(0, 0);
         window.setLayout(new BorderLayout());
         FilterFrame main = new FilterFrame();
         final JFrame center = new JFrame();
         main.add(center, BorderLayout.CENTER);
         main.add(new JToolBar.Separator(), BorderLayout.NORTH);
```

```
main.add(new JToolBar.Separator(), BorderLayout.WEST);
main.add(new JToolBar.Separator(), BorderLayout.EAST);
center.setLayout(new GridLayout(5, 1, 10, 10));
JTextField field1 = new JTextField("БАКАЛАВРСКАЯ РАБОТА НА ТЕМУ:");
field1.setFont(timesNewRomanFont14b);
field1.setHorizontalAlignment(0);
center.add(field1);
JTextField field2 = new JTextField("Измерение объемного расхода газа в малых и сверхмалых объемах");
field2.setFont(timesNewRomanFont12);
field2.setHorizontalAlignment(0);
center.add(field2);
JTextField field3 = new JTextField("Автор: Лавринов С.В.");
field3.setHorizontalAlignment(0);
center.add(field3);
JTextField field4 = new JTextField("Руководитель: Кюрджиев Ю.В.");
field4.setHorizontalAlignment(0);
center.add(field4);
JButton startButton = new JButton("Старт");
startButton.setHorizontalAlignment(0);
startButton.setSize(100, 12);
startButton.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent evt) {
      try {
         isCapturingVideo = true;
         center.setVisible(false);
         //gets a list of devices how support the given videoformat
         Vector deviceList = CaptureDeviceManager.getDeviceList(new YUVFormat());
         device = (CaptureDeviceInfo) deviceList.firstElement();
         System.out.println("Device: " + device);
         ml = device.getLocator();
         player = Manager.createRealizedPlayer(ml);
         player.start();
         fgc = (FrameGrabbingControl) player.getControl("javax.media.control.FrameGrabbingControl");
         bti = new BufferToImage((VideoFormat) (fgc.grabFrame()).getFormat());
         javax.swing.Timer tu = new javax.swing.Timer(30, new Updater());
         tu.start();
         javax.swing.Timer fpsT = new javax.swing.Timer(1000, new FPS_Listener());
         fpsT.start();
      } catch (Exception exc) {
         exc.printStackTrace();
      }
   }
});

window.add(main, BorderLayout.WEST);

JPanel east = new JPanel();
main.get(BorderLayout.EAST).setLayout(new GridLayout(5, 1));

javax.swing.JCheckBox cbRadiusDiff = new JCheckBox("Radius Method");
cbRadiusDiff.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent evt) {

      serialMoveMethod = serialMoveMethod == RADIUS ? -1 : RADIUS;
   }
});
east.add(cbRadiusDiff);
javax.swing.JCheckBox cbXSpot = new JCheckBox("XSpot Method");
cbXSpot.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent evt) {

      serialMoveMethod = serialMoveMethod == XSPOT ? -1 : XSPOT;
   }
});
east.add(cbXSpot);
```

```
            window.setVisible(true);

        } catch (Exception e) {
            System.out.println(e);
            e.printStackTrace();
        }
    }

    public class Updater implements ActionListener {
        public void actionPerformed(ActionEvent evt) {
            pPanel.repaint();
        }
    }

    public Buffer buf;
    public Image img;
    public BufferToImage bti;

    public int[][] sobelX = {{3, 10, 3}, {0, 0, 0}, {-3, -10, -3}};
    public int[][] sobelY = {{3, 0, -3}, {10, 0, -10}, {3, 0, -3}};
    public int[][] roiMap, refinedROImap;
    public int[][][] sobelMap;

    public int roiBasis;
    public double roiDeviation = .20;
    public HoughFilter hf = new HoughFilter(null, 15, 25, null);
    public int lastRadius;
    public int preferredRegion;
    public int lastCenterX, lastCenterY, lastDrawYBasis;
    public int radBegin, radEnd, radIncrement;
    public int consecutiveNoFinds;
    public int guessMinX, guessMaxX, guessMinY, guessMaxY;

    public static final int TOP = 0, BOTTOM = 1, CENTER = 2, ALL = 3, COMPRESS = 4;
    public static final String[] regionList = {"TOP", "BOTTOM", "CENTER", "ALL", "COMPRESS"};
    public CED_fast canny;
    public BufferedImage cannyImage;

    public class FilterFrame extends JPanel {
        public FilterFrame() {
            super();
            preferredRegion = BOTTOM;
            lastCenterX = -1;
            lastCenterY = -1;
            lastRadius = -1;
            lastDrawYBasis = 0;
            consecutiveNoFinds = 0;
            canny = new CED_fast();
        }

        public void paintComponent(Graphics g) {
            super.paintComponent(g);

            if (!isCapturingVideo) {
                g.setColor(Color.black);
                g.drawString("Добро пожаловать!", 10, 20);
                g.drawString("Данная программа предназначена для определения массового и объемного расхода газа в малых
и сверхмалых процессах", 10, 20);
                g.drawString("Программа анализирует область и выделяет сферические объекты, которые являются пузырьками
газа", 20, 20);
                g.drawString("и определяя радиус объектов вычисляет суммарный расход газа, проходящий через
измерительное устройство ", 30, 20);
                g.drawString("Нажмите на кнопку старт", 10, 20);
                return;
            }
            buf = fgc.grabFrame();
```

```
        img = (new BufferToImage((VideoFormat) buf.getFormat()).createImage(buf));

        g.drawImage(img, 0, 0, null);

        int w = img.getWidth(null);
        int h = img.getHeight(null);
        int avgcol = 0;
        int cts = 0;

        int[] pixels = new int[img.getWidth(null) * img.getHeight(null)];
        int[] pixels_all = new int[img.getWidth(null) * img.getHeight(null)];
        int[] pixels_region2 = new int[img.getWidth(null) * img.getHeight(null)];
        int[] pixels_region3 = new int[img.getWidth(null) * img.getHeight(null)];
        int[] pixels_compress = new int[img.getWidth(null) * img.getHeight(null)];
        int[][] map = new int[h / 2][w / 2];
        roiMap = new int[h / 2][w / 2];
        refinedROImap = new int[h / 2][w / 2];
        int[][] hitMap = new int[h / 2][w / 2];
        PixelGrabber pg = new PixelGrabber(img, 0, 0, w, h, pixels, 0, w);
        int reg3index = 0;
        int reg2index = 0;
        int reg1index = 0;
        int reg1index_compress = 0;
        try {
          pg.grabPixels();
          int increment = 1;
          for (int j = 0; j < h; j += increment) {
            for (int i = 0; i < w; i += increment) {
              //int alpha = (pixels[i] >> 24) & 0xff;
              float red = (pixels[i + j * w] >> 16) & 0xff;
              float green = (pixels[i + j * w] >> 8) & 0xff;
              float blue = (pixels[i + j * w]) & 0xff;
              int gray = (int) (red * .3 + green * .59 + blue * .11);
              map[j / 2][i / 2] = gray;
              hitMap[j / 2][i / 2] = 0;
              if ((j % 2) == 0 && (i % 2) == 0) {
                pixels_compress[reg1index_compress] = Math.round(0.299f * red + 0.587f * green + 0.114f * blue);
                reg1index_compress++;
              }

              pixels_all[j / increment * w / increment + i / increment] = Math.round(0.299f * red + 0.587f * green + 0.114f *
blue);
              //pixels[j/increment * w/increment + i/increment] = Math.round(0.299f * red + 0.587f * green + 0.114f * blue);
              if (j < h / 2) {
                pixels_region2[reg2index] = Math.round(0.299f * red + 0.587f * green + 0.114f * blue);
                reg2index++;
              } else {
                pixels_region3[reg3index] = Math.round(0.299f * red + 0.587f * green + 0.114f * blue);
                reg3index++;
              }
              if (j > h / 4 && j < 3 * h / 4) {
                pixels[reg1index] = Math.round(0.299f * red + 0.587f * green + 0.114f * blue);
                reg1index++;
              }

            }
          }
          sobelMap = sobelMapping(map);
          int averageVal = fiveRegions(map);
          for (int j = h / 2 - 1; j >= 0; j--) {
            points = 0;
            recursiveCatch(map, sobelMap, hitMap, averageVal, w / 4, j);
          }
          Polygon[] gons = createPolygon(hitMap);
          g.setColor(Color.yellow);
          g.drawPolygon(gons[1]);
```

```
        int drawY_basis = 0;
        switch (preferredRegion) {
          case ALL:
            canny.process_fast(pixels_all, w, h);
            radBegin = 10;
            radEnd = 80;
            radIncrement = 10;
            drawY_basis = 0;

            break;
          case TOP:
            canny.process_fast(pixels_region2, w, h / 2);
            radBegin = 10;
            radEnd = 26;
            radIncrement = 4;
            drawY_basis = 0;
            break;
          case BOTTOM:
            canny.process_fast(pixels_region3, w, h / 2);
            radBegin = 45;
            radEnd = 80;
            radIncrement = 5;
            drawY_basis = h / 2;
            break;
          case CENTER:
            canny.process_fast(pixels, w, h / 2);
            radBegin = 25;
            radEnd = 45;
            radIncrement = 5;
            drawY_basis = h / 4;
            break;
          case COMPRESS:
            canny.process_fast(pixels_compress, w / 2, h / 2);
            radBegin = 5;
            radEnd = 40;
            radIncrement = 5;
            drawY_basis = 0;
            break;
          default:
            break;
        }
        cannyImage = canny.getEdgesImage();
        g.drawImage(cannyImage, 0, img.getHeight(null), null);

        int bestAccumulatorValue = 0;
        int bestLocationX = 0;
        int bestLocationY = 0;
        int bestRadius = 0;
        int maxx = canny.getWidth();
        int maxy = canny.getHeight();
        int hf_width = canny.getWidth();
        int hf_height = canny.getHeight();

        guessMinX = Math.max(0, lastCenterX - 3 * lastRadius);
        guessMaxX = Math.min(w, lastCenterX + 3 * lastRadius);
        guessMinY = Math.max(0, lastCenterY - 3 * lastRadius);
        guessMaxY = Math.min(h, lastCenterY + 3 * lastRadius);

        int[] scaledRegion = new int[]{guessMinX, Math.max(0, lastCenterY - 3 * lastRadius - drawY_basis), guessMaxX,
Math.min(hf_height, lastCenterY + 3 * lastRadius - drawY_basis)};

        g.setColor(Color.orange);
        g.drawRect(guessMinX, guessMinY, guessMaxX - guessMinX, guessMaxY - guessMinY);
        g.setColor(Color.green);
        g.drawRect(scaledRegion[0], scaledRegion[1] + img.getHeight(null), scaledRegion[2] - scaledRegion[0], scaledRegion[3]
- scaledRegion[1]);
```

```
        int[] hf_region = new int[]{scaledRegion[0], scaledRegion[1], scaledRegion[2] - scaledRegion[0], scaledRegion[3] -
scaledRegion[1]};

        if (preferredRegion == COMPRESS) hf_region = new int[]{0, 0, maxx, maxy};
        hf.setRegion(hf_region);
        hf.createSourceArray(canny.getData(), hf_width, hf_height);
        for (int rad = radBegin; rad <= radEnd; rad += radIncrement) {
          hf.setParams(rad, rad + 7);
          hf.resetChangeableArrays(hf_region);
          hf.process(false);
          java.util.LinkedList<Point> ps = hf.getPossibilityList();
          for (int i = 0; i < ps.size(); i++) {
            Point p = ps.get(i);
            double drawX = p.getX();
            double drawY = p.getY() + drawY_basis;
            if (preferredRegion == COMPRESS) {
              drawX = drawX * 2;
              drawY = (drawY - drawY_basis) * 2 + drawY_basis;
            }
            if (hf.valList.get(i) > bestAccumulatorValue && gons[1].contains(drawX, drawY)) {
              bestLocationX = (int) drawX;
              bestLocationY = (int) drawY;
              bestRadius = rad;
              bestAccumulatorValue = hf.valList.get(i);
            }
          }
        }
        boolean bubbleFound = true;
        if (bestRadius > 0) {
          if (preferredRegion == COMPRESS) {
            lastRadius = bestRadius * 2;
            lastCenterX = bestLocationX;
            lastCenterY = bestLocationY;
            lastDrawYBasis = drawY_basis;
            consecutiveNoFinds = 0;
          } else {
            lastRadius = bestRadius;
            lastCenterX = bestLocationX;
            lastCenterY = bestLocationY;
            lastDrawYBasis = drawY_basis;
            consecutiveNoFinds = 0;
          }

          if (preferredRegion != COMPRESS)
            if (lastCenterY < 2 * h / 5) {
              preferredRegion = TOP;
            } else if (lastCenterY > 3 * h / 5) {
              preferredRegion = BOTTOM;
            } else
              preferredRegion = CENTER;

        } else {
          preferredRegion = (preferredRegion == COMPRESS) ? ALL : COMPRESS; //COMPRESS;
          consecutiveNoFinds++;

        }
        int yDist = (int) (-0.46667 * (double) lastRadius + 34.6667);
        if (consecutiveNoFinds < 3) {
          g.setColor(Color.blue);
          g.drawOval((int) lastCenterX - lastRadius, (int) lastCenterY - lastRadius, 2 * lastRadius, 2 * lastRadius);

          int cameraX = cameraPosition * 2;
          if (trackBubble) {
            if (lastCenterX > 170) {
              int difference = 0;
              if (serialMoveMethod == RADIUS)
```

```
                difference = (int) ((double) (lastCenterX - 160) / (double) lastRadius * 2.25);
              if (serialMoveMethod == XSPOT)
                difference = (int) ((double) (lastCenterX - 160) / 2);

              cameraPosition += difference;
              if (cameraPosition > maxCameraPosition) cameraPosition = maxCameraPosition;
            }
            if (lastCenterX < 150) {
              int difference = 0;
              if (serialMoveMethod == RADIUS)
                difference = (int) ((double) (lastCenterX - 160) / (double) lastRadius * 2.25);

              cameraPosition += difference;
              if (cameraPosition < 10) cameraPosition = 10;
            }
          }
        } else {
          g.setColor(Color.red);
          g.drawString("не найден!!!", img.getWidth(null) + 20, img.getHeight(null));
          lastCenterX = img.getWidth(null) / 2;
          lastCenterY = img.getHeight(null) / 2;
          lastRadius = img.getWidth(null);
          bubbleFound = false;
          //cameraPosition = (char)80;
          if (trackBubble) {
            if (cameraPosition > 80) cameraPosition -= 2;
            if (cameraPosition < 70) cameraPosition += 2;
            //todo
            System.out.println("cameraPosition = " + cameraPosition);
          }
        }
        currFPS++;
        g.setColor(Color.red);
        g.drawString("FPS: " + lastFPS, 10, 20);
        g.setColor(Color.black);
        g.drawString("Регион: " + regionList[preferredRegion], img.getWidth(null) + 10, 20);
        g.drawString("Радиус пузырька: " + bestRadius, img.getWidth(null) + 10, 35);
        g.drawString("Позиция: (" + bestLocationX + " , " + bestLocationY + ")", img.getWidth(null) + 10, 50);
        g.drawString("Аккумулятор: " + bestAccumulatorValue, img.getWidth(null) + 10, 65);
        g.drawString("Позиция камеры: " + (byte) cameraPosition, img.getWidth(null) + 10, 80);
        g.drawString("---------- ", img.getWidth(null) + 10, 80);
        g.drawString("Объемный Расход: " + (byte) cameraPosition, img.getWidth(null) + 10, 80);
      } catch (Exception exc) {
        System.out.println(exc);
        exc.printStackTrace();
      }
    }
  }

  public int fiveRegions(int[][] grayImage) {
    int w = grayImage[0].length;
    int h = grayImage.length;
    int avgMid = 0, avgLeft = 0, avgRight = 0, avgUp = 0, avgDown = 0;

    for (int j = h / 2 - 3; j <= h / 2 + 3; j++) {
      for (int i = w / 2 - 3; i <= w / 2 + 3; i++) {
        avgMid += grayImage[j][i];
      }

      for (int i = w / 3 - 3; i <= w / 3 + 3; i++) {
        avgLeft += grayImage[j][i];
      }

      for (int i = 2 * w / 3 - 3; i <= 2 * w / 3 + 3; i++) {
        avgRight += grayImage[j][i];
      }
```

```
    }

    for (int i = w / 2 - 3; i <= w / 2 + 3; i++) {
      for (int j = h / 3 - 3; j <= h / 3 + 3; j++) {
        avgUp += grayImage[j][i];
      }

      for (int j = 2 * h / 3 - 3; j <= 2 * h / 3 + 3; j++) {
        avgDown += grayImage[j][i];
      }
    }
    int min = Math.min(avgUp, Math.min(avgDown, Math.min(avgLeft, Math.min(avgRight, avgMid))));
    int max = Math.max(avgUp, Math.max(avgDown, Math.max(avgLeft, Math.max(avgRight, avgMid))));
    int avg = ((avgUp + avgDown + avgLeft + avgRight + avgMid) - (min + max)) / 3 / 49;

    return avg;
  }

  public void findBlob(int[][] redMap, int[][] hitMap, int i, int j, int hn) {
    if (redMap[j][i] == 1 && hitMap[j][i] == 0) hitMap[j][i] = hn;
    else return;
    if (i + 1 < redMap[0].length) findBlob(redMap, hitMap, i + 1, j, hn); //right
    if (i - 1 >= 0) findBlob(redMap, hitMap, i - 1, j, hn); //left
    if (j + 1 < redMap.length) findBlob(redMap, hitMap, i, j + 1, hn); //down
    if (j - 1 >= 0) findBlob(redMap, hitMap, i, j - 1, hn); //up

    return;
  }

  public int points = 0;

  public Polygon[] createPolygon(int[][] hitmap) {
    Polygon toReturn = new Polygon();
    Polygon toReturn2 = new Polygon();
    int num_in_required = 4;
    for (int j = hitmap.length - 1; j >= 0; j--) {
      int inwards = 0;
      for (int i = 0; i < 2 * hitmap[0].length / 3; i++) {
        if (hitmap[j][i] == 1) {
          inwards++;
          if (inwards == num_in_required) {
            toReturn.addPoint(i, j);
            toReturn2.addPoint(i * 2, j * 2);
            break;
          }
        }
      }
    }
    for (int j = 0; j < hitmap.length; j++) {
      int inwards = 0;
      for (int i = hitmap[0].length - 1; i > hitmap[0].length / 3; i--) {

        if (hitmap[j][i] == 1) {
          inwards++;
          if (inwards == num_in_required) {
            toReturn.addPoint(i, j);
            toReturn2.addPoint(i * 2, j * 2);
            break;
          }
        }
      }
    }
    return new Polygon[]{toReturn, toReturn2};
  }

  public void recursiveCatch(int[][] map, int[][][] gradient, int[][] hitMap, int avg, int i, int j) {
```

```
    if (points > 6000) return;
    points++;
    if (hitMap[j][i] == 0) {
      if (map[j][i] <= (double) avg * 1.25 && Math.abs(gradient[0][j][i]) < 200 && Math.abs(gradient[1][j][i]) < 200) {
        hitMap[j][i] = 1;
        if (i + 1 < map[0].length) recursiveCatch(map, gradient, hitMap, avg, i + 1, j);
        if (j - 1 >= 0) recursiveCatch(map, gradient, hitMap, avg, i, j - 1);
        if (i - 1 >= 0) recursiveCatch(map, gradient, hitMap, avg, i - 1, j);
        if (j + 1 < map.length) recursiveCatch(map, gradient, hitMap, avg, i, j + 1);
      } else {
        if (Math.abs(gradient[0][j][i]) < 120 && Math.abs(gradient[1][j][i]) < 120 && map[j][i] <= (double) avg * 1.4) {
          hitMap[j][i] = 1;

          if (i + 1 < map[0].length) recursiveCatch(map, gradient, hitMap, avg, i + 1, j);
          if (i - 1 >= 0) recursiveCatch(map, gradient, hitMap, avg, i - 1, j);
          if (j + 1 < map.length) recursiveCatch(map, gradient, hitMap, avg, i, j + 1);
          if (j - 1 >= 0) recursiveCatch(map, gradient, hitMap, avg, i, j - 1);
        } else
          return;
      }
    } else {
      return;
    }
  }

  public void tryRecur(int[][] map, int[][] sobel, int[][] out, int i, int j, int[][] hitmap) {
    if (i < map[0].length && i >= 0 && j >= 0 && j < map.length && hitmap[j][i] == 0) {
      out[j][i] = 0;

      if (map[j][i] <= roiBasis + (int) ((double) roiBasis * roiDeviation)) {
        out[j][i] = 1;
        hitmap[j][i] = 1;
        tryRecur(map, sobel, out, i + 1, j, hitmap);//right
        tryRecur(map, sobel, out, i - 1, j, hitmap);//left
        tryRecur(map, sobel, out, i, j + 1, hitmap); //down
        tryRecur(map, sobel, out, i, j - 1, hitmap);//up

      } else if (sobel[j][i] <= 200) {
        hitmap[j][i] = 1;
        out[j][i] = 1;
        tryRecur(map, sobel, out, i + 1, j, hitmap);//right
        tryRecur(map, sobel, out, i - 1, j, hitmap);//left
        tryRecur(map, sobel, out, i, j + 1, hitmap); //down
        tryRecur(map, sobel, out, i, j - 1, hitmap);//up
      }
    }
    return;
  }

 /* public int[] blobArea(int[][] hitMap, int hn) {
    int[] res = new int[hn];
    for (int j = 0; j < hitMap.length; j++) {
      for (int i = 0; i < hitMap[0].length; i++) {
        res[hitMap[j][i]]++;
      }
    }
    return res;
  }*/

  public int[][][] sobelMapping(int[][] grayImage) {
    int[][][] result = new int[2][grayImage.length][grayImage[0].length];
    //int[][] resultY = new int[grayImage.length][grayImage[0].length];

    for (int j = 0; j < grayImage.length; j++) {
      for (int i = 0; i < grayImage[0].length; i++) {
```

```
            if (!(j - 1 < 0 || j + 1 >= grayImage.length
                  || i - 1 < 0 || i + 1 >= grayImage[0].length)) {
               result[0][j][i] = (grayImage[j - 1][i - 1] * sobelX[0][0] +
                     grayImage[j - 1][i] * sobelX[0][1] +
                     grayImage[j - 1][i + 1] * sobelX[0][2] +
                     grayImage[j][i - 1] * sobelX[1][0] +
                     grayImage[j][i] * sobelX[1][1] +
                     grayImage[j][i + 1] * sobelX[1][2] +
                     grayImage[j + 1][i - 1] * sobelX[2][0] +
                     grayImage[j + 1][i] * sobelX[2][1] +
                     grayImage[j + 1][i + 1] * sobelX[2][2]);

               result[1][j][i] = (grayImage[j - 1][i - 1] * sobelY[0][0] +
                     grayImage[j - 1][i] * sobelY[0][1] +
                     grayImage[j - 1][i + 1] * sobelY[0][2] +
                     grayImage[j][i - 1] * sobelY[1][0] +
                     grayImage[j][i] * sobelY[1][1] +
                     grayImage[j][i + 1] * sobelY[1][2] +
                     grayImage[j + 1][i - 1] * sobelY[2][0] +
                     grayImage[j + 1][i] * sobelY[2][1] +
                     grayImage[j + 1][i + 1] * sobelY[2][2]);
            } else {
               result[0][j][i] = 0;
               result[1][j][i] = 0;
            }
         }
      }

      return result;
   }

   public class FPS_Listener implements ActionListener {
      public void actionPerformed(ActionEvent evt) {
         lastFPS = currFPS;
         currFPS = 0;
      }
   }

   private static void getList() {
      // TODO Auto-generated method stub
      Vector devices =
            (Vector) CaptureDeviceManager.getDeviceList(null);
      System.out.println(devices.size());
      Enumeration enumz = devices.elements();
      while (enumz.hasMoreElements()) {
         CaptureDeviceInfo cdi = (CaptureDeviceInfo) enumz.nextElement();
         String name = cdi.getName();
         javax.media.Format[] fmts = cdi.getFormats();
         System.out.println(name);
         for (int i = 0; i < fmts.length; i++) {
            System.out.println(fmts[i]);
            if (name.startsWith("vfw:")) {
               System.out.println("" + fmts[i]);
            }
         }
      }
   }
}
```

Класс CED_fast.java

```
import java.awt.image.BufferedImage;
```

```
import java.util.Arrays;


public class CED_fast {

   // statics

   private final static float GAUSSIAN_CUT_OFF = 0.005f;
   private final static float MAGNITUDE_SCALE = 100F;
   private final static float MAGNITUDE_LIMIT = 1000F;
   private final static int MAGNITUDE_MAX = (int) (MAGNITUDE_SCALE * MAGNITUDE_LIMIT);

   // fields

   private int height;
   private int width;
   private int picsize;
   private int[] data;
   private int[] magnitude;
   private BufferedImage sourceImage;
   private BufferedImage edgesImage;

   private float gaussianKernelRadius;
   private float lowThreshold;
   private float highThreshold;
   private int gaussianKernelWidth;
   private boolean contrastNormalized;

   private float[] xConv;
   private float[] yConv;
   private float[] xGradient;
   private float[] yGradient;


   public CED_fast() {
      lowThreshold = 2.5f;
      highThreshold = 7.5f;
      gaussianKernelRadius = 2f;
      gaussianKernelWidth = 16;
      contrastNormalized = false;
      this.createKernels(gaussianKernelRadius, gaussianKernelWidth);
   }

   public int[] getData() {
      return data;
   }

   public int getWidth() {
      return width;

   }

   public int getHeight() {
      return height;
   }

   //read in the pixels from the pixel grabber
   public void setData(int[] d) {
      data = d;
   }


   public BufferedImage getSourceImage() {
      return sourceImage;
   }
```

```
public void setSourceImage(BufferedImage image) {
   sourceImage = image;
}


public BufferedImage getEdgesImage() {
   return edgesImage;
}


public void setEdgesImage(BufferedImage edgesImage) {
   this.edgesImage = edgesImage;
}


public float getLowThreshold() {
   return lowThreshold;
}


public void setLowThreshold(float threshold) {
   if (threshold < 0) throw new IllegalArgumentException();
   lowThreshold = threshold;
}


public float getHighThreshold() {
   return highThreshold;
}


public void setHighThreshold(float threshold) {
   if (threshold < 0) throw new IllegalArgumentException();
   highThreshold = threshold;
}


public int getGaussianKernelWidth() {
   return gaussianKernelWidth;
}


public void setGaussianKernelWidth(int gaussianKernelWidth) {
   if (gaussianKernelWidth < 2) throw new IllegalArgumentException();
   this.gaussianKernelWidth = gaussianKernelWidth;
}


public float getGaussianKernelRadius() {
   return gaussianKernelRadius;
}


public void setGaussianKernelRadius(float gaussianKernelRadius) {
   if (gaussianKernelRadius < 0.1f) throw new IllegalArgumentException();
   this.gaussianKernelRadius = gaussianKernelRadius;
}


public boolean isContrastNormalized() {
   return contrastNormalized;
}


public void setContrastNormalized(boolean contrastNormalized) {
```

```
    this.contrastNormalized = contrastNormalized;
}


public void process() {
    width = sourceImage.getWidth();
    height = sourceImage.getHeight();
    picsize = width * height;
    initArrays();
    readLuminance();
    if (contrastNormalized) normalizeContrast();
    computeGradients(gaussianKernelRadius, gaussianKernelWidth);
    int low = Math.round(lowThreshold * MAGNITUDE_SCALE);
    int high = Math.round(highThreshold * MAGNITUDE_SCALE);
    performHysteresis(low, high);
    thresholdEdges();
    writeEdges(data);
}

public void process_fast(int[] data_in, int w, int h) {
    width = w;
    height = h;
    picsize = width * height;
    initArrays();
    readLuminance_fast(data_in);
    if (contrastNormalized) normalizeContrast();
    computeGradients(gaussianKernelRadius, gaussianKernelWidth);
    int low = Math.round(lowThreshold * MAGNITUDE_SCALE);
    int high = Math.round(highThreshold * MAGNITUDE_SCALE);
    performHysteresis(low, high);
    thresholdEdges();
    writeEdges(data);
}

private void initArrays() {
    //if (data == null || picsize != data.length) {
    data = new int[picsize];
    magnitude = new int[picsize];

    xConv = new float[picsize];
    yConv = new float[picsize];
    xGradient = new float[picsize];
    yGradient = new float[picsize];
    //}
}

public float kernel[];
public float diffKernel[];

public void createKernels(float kernelRadius, int kernelWidth) {
    kernel = new float[kernelWidth];
    diffKernel = new float[kernelWidth];
    int kwidth;
    for (kwidth = 0; kwidth < kernelWidth; kwidth++) {
        float g1 = gaussian(kwidth, kernelRadius);
        if (g1 <= GAUSSIAN_CUT_OFF && kwidth >= 2) break;
        float g2 = gaussian(kwidth - 0.5f, kernelRadius);
        float g3 = gaussian(kwidth + 0.5f, kernelRadius);
        kernel[kwidth] = (g1 + g2 + g3) / 3f / (2f * (float) Math.PI * kernelRadius * kernelRadius);
        diffKernel[kwidth] = g3 - g2;
    }
}

private void computeGradients(float kernelRadius, int kernelWidth) {
    //int initX = kwidth - 1;
    int initX = kernelWidth - 1;
```

```
int maxX = width - (kernelWidth - 1);
int initY = width * (kernelWidth - 1);
int maxY = width * (height - (kernelWidth - 1));
//perform convolution in x and y directions
for (int x = initX; x < maxX; x++) {
   for (int y = initY; y < maxY; y += width) {
      int index = x + y;
      float sumX = data[index] * kernel[0];
      float sumY = sumX;
      int xOffset = 1;
      int yOffset = width;
      for (; xOffset < kernelWidth; ) {
         sumY += kernel[xOffset] * (data[index - yOffset] + data[index + yOffset]);
         sumX += kernel[xOffset] * (data[index - xOffset] + data[index + xOffset]);
         yOffset += width;
         xOffset++;
      }
      //System.out.println("index = " + index + " yconv.length: " + yConv.length);
      yConv[index] = sumY;
      xConv[index] = sumX;
   }
}

for (int x = initX; x < maxX; x++) {
   for (int y = initY; y < maxY; y += width) {
      float sum = 0f;
      int index = x + y;
      for (int i = 1; i < kernelWidth; i++)
         sum += diffKernel[i] * (yConv[index - i] - yConv[index + i]);

      xGradient[index] = sum;
   }
}

for (int x = kernelWidth; x < width - kernelWidth; x++) {
   for (int y = initY; y < maxY; y += width) {
      float sum = 0.0f;
      //float sum2 = 0.0f;
      int index = x + y;
      //int index2 = (x-1) + y;
      int yOffset = width;
      for (int i = 1; i < kernelWidth; i++) {
         sum += diffKernel[i] * (xConv[index - yOffset] - xConv[index + yOffset]);
         //sum2 += diffKernel[i] * (yConv[index - i] - yConv[index + i]);
         yOffset += width;
      }
      yGradient[index] = sum;
   }
}
initX = kernelWidth;
maxX = width - kernelWidth;
initY = width * kernelWidth;
maxY = width * (height - kernelWidth);
for (int x = initX; x < maxX; x++) {
   for (int y = initY; y < maxY; y += width) {
      int index = x + y;
      int indexN = index - width;
      int indexS = index + width;
      int indexW = index - 1;
      int indexE = index + 1;
      int indexNW = indexN - 1;
      int indexNE = indexN + 1;
      int indexSW = indexS - 1;
      int indexSE = indexS + 1;

      float xGrad = xGradient[index];
```

```
            float yGrad = yGradient[index];
            float gradMag = hypot(xGrad, yGrad);
            //perform non-maximal supression
            float nMag = hypot(xGradient[indexN], yGradient[indexN]);
            float sMag = hypot(xGradient[indexS], yGradient[indexS]);
            float wMag = hypot(xGradient[indexW], yGradient[indexW]);
            float eMag = hypot(xGradient[indexE], yGradient[indexE]);
            float neMag = hypot(xGradient[indexNE], yGradient[indexNE]);
            float seMag = hypot(xGradient[indexSE], yGradient[indexSE]);
            float swMag = hypot(xGradient[indexSW], yGradient[indexSW]);
            float nwMag = hypot(xGradient[indexNW], yGradient[indexNW]);
            float tmp;
            if (xGrad * yGrad <= (float) 0 /*(1)*/
                  ? Math.abs(xGrad) >= Math.abs(yGrad) /*(2)*/
                  ? (tmp = Math.abs(xGrad * gradMag)) >= Math.abs(yGrad * neMag - (xGrad + yGrad) * eMag) /*(3)*/
                  && tmp > Math.abs(yGrad * swMag - (xGrad + yGrad) * wMag) /*(4)*/
                  : (tmp = Math.abs(yGrad * gradMag)) >= Math.abs(xGrad * neMag - (yGrad + xGrad) * nMag) /*(3)*/
                  && tmp > Math.abs(xGrad * swMag - (yGrad + xGrad) * sMag) /*(4)*/
                  : Math.abs(xGrad) >= Math.abs(yGrad) /*(2)*/
                  ? (tmp = Math.abs(xGrad * gradMag)) >= Math.abs(yGrad * seMag + (xGrad - yGrad) * eMag) /*(3)*/
                  && tmp > Math.abs(yGrad * nwMag + (xGrad - yGrad) * wMag) /*(4)*/
                  : (tmp = Math.abs(yGrad * gradMag)) >= Math.abs(xGrad * seMag + (yGrad - xGrad) * sMag) /*(3)*/
                  && tmp > Math.abs(xGrad * nwMag + (yGrad - xGrad) * nMag) /*(4)*/
                  ) {
               magnitude[index] = gradMag >= MAGNITUDE_LIMIT ? MAGNITUDE_MAX : (int) (MAGNITUDE_SCALE * gradMag);

            } else {
               magnitude[index] = 0;
            }
         }
      }
   }

   private float hypot(float x, float y) {
      if (x == 0f) return y;
      if (y == 0f) return x;
      return (float) Math.sqrt(x * x + y * y);
   }

   private float gaussian(float x, float sigma) {
      return (float) Math.exp(-(x * x) / (2f * sigma * sigma));
   }

   private void performHysteresis(int low, int high) {
      Arrays.fill(data, 0);
      int offset = 0;
      for (int x = 0; x < width; x++) {
         for (int y = 0; y < height; y++) {
            if (data[offset] == 0 && magnitude[offset] >= high) {
               follow(x, y, offset, low);
            }
            offset++;
         }
      }
   }

   private void follow(int x1, int y1, int i1, int threshold) {
      int x0 = x1 == 0 ? x1 : x1 - 1;
      int x2 = x1 == width - 1 ? x1 : x1 + 1;
      int y0 = y1 == 0 ? y1 : y1 - 1;
      int y2 = y1 == height - 1 ? y1 : y1 + 1;

      data[i1] = magnitude[i1];
      for (int x = x0; x <= x2; x++) {
         for (int y = y0; y <= y2; y++) {
            int i2 = x + y * width;
```

```
            if ((y != y1 || x != x1)
                  && data[i2] == 0
                  && magnitude[i2] >= threshold) {
               follow(x, y, i2, threshold);
               return;
            }
         }
      }
   }

   private void thresholdEdges() {
      for (int i = 0; i < picsize; i++) {
         data[i] = data[i] > 0 ? -1 : 0xff000000;

      }
   }

   private int luminance(float r, float g, float b) {
      return Math.round(0.299f * r + 0.587f * g + 0.114f * b);
   }

   private void readLuminance_fast(int[] data_in) {
      data = data_in;
   }

   private void readLuminance() {
      int type = sourceImage.getType();
      if (type == BufferedImage.TYPE_INT_RGB || type == BufferedImage.TYPE_INT_ARGB) {
         int[] pixels = (int[]) sourceImage.getData().getDataElements(0, 0, width, height, null);
         for (int i = 0; i < picsize; i++) {
            int p = pixels[i];
            int r = (p & 0xff0000) >> 16;
            int g = (p & 0xff00) >> 8;
            int b = p & 0xff;
            data[i] = luminance(r, g, b);
         }
      } else if (type == BufferedImage.TYPE_BYTE_GRAY) {
         byte[] pixels = (byte[]) sourceImage.getData().getDataElements(0, 0, width, height, null);
         for (int i = 0; i < picsize; i++) {
            data[i] = (pixels[i] & 0xff);
         }
      } else if (type == BufferedImage.TYPE_USHORT_GRAY) {
         short[] pixels = (short[]) sourceImage.getData().getDataElements(0, 0, width, height, null);
         for (int i = 0; i < picsize; i++) {
            data[i] = (pixels[i] & 0xffff) / 256;
         }
      } else if (type == BufferedImage.TYPE_3BYTE_BGR) {
         byte[] pixels = (byte[]) sourceImage.getData().getDataElements(0, 0, width, height, null);
         int offset = 0;
         for (int i = 0; i < picsize; i++) {
            int b = pixels[offset++] & 0xff;
            int g = pixels[offset++] & 0xff;
            int r = pixels[offset++] & 0xff;
            data[i] = luminance(r, g, b);
         }
      } else {
         throw new IllegalArgumentException("Unsupported image type: " + type);
      }
   }

   private void normalizeContrast() {
      int[] histogram = new int[256];
      for (int i = 0; i < data.length; i++) {
         histogram[data[i]]++;
      }
      int[] remap = new int[256];
```

```
        int sum = 0;
        int j = 0;
        for (int i = 0; i < histogram.length; i++) {
            sum += histogram[i];
            int target = sum * 255 / picsize;
            for (int k = j + 1; k <= target; k++) {
                remap[k] = i;
            }
            j = target;
        }
        for (int i = 0; i < data.length; i++) {
            data[i] = remap[data[i]];
        }
    }
    private void writeEdges(int pixels[]) {
        //if (edgesImage == null) {
        edgesImage = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);
        //}
        edgesImage.getWritableTile(0, 0).setDataElements(0, 0, width, height, pixels);
    }
}
```

Класс HoughFilter.java

```
/*
* Lavrinov Sergey 10-10-2015
* */

import java.awt.image.*;
import java.util.*;
import java.awt.*;

public class HoughFilter {

    public BufferedImage sourceImage;
    public int[][] accumulator, regionalMax;
    public int radius, threshold;
    public int[] region;
    public int[][] sourceArray;
    public Raster raster;
    public LinkedList<Integer> xlist;
    public LinkedList<Integer> ylist;
    public int height, width;
    public LinkedList<Point> possibilityList;
    public int[] masterData;
    public int[][] zeroAccumulator;

    public HoughFilter(BufferedImage img, int r, int thresh, int[] reg) {
        if (img != null) {
            sourceImage = img;
            raster = sourceImage.getData();
            accumulator = new int[img.getHeight()][img.getWidth()];
            height = img.getHeight();
            width = img.getWidth();
        }
        radius = r;
        threshold = thresh;
        region = reg;
        xlist = new LinkedList<Integer>();
        ylist = new LinkedList<Integer>();
        //createArray();
    }

    public HoughFilter(int[] data, int r, int thresh, int[] reg, int w, int h) {
        xlist = new LinkedList<Integer>();
        ylist = new LinkedList<Integer>();
```

```
    sourceArray = new int[h][w];
    for (int j = 0; j < h; j++) {
      for (int i = 0; i < w; i++) {
        sourceArray[j][i] = (data[j * w + i] != 0) ? 255 : 0;
        if (sourceArray[j][i] == 255) {
          sourceArray[j][i] = 1;
          xlist.add(i);
          ylist.add(j);
        }
        accumulator[j][i] = 0;
      }
    }
    radius = r;
    threshold = thresh;
    region = reg;
    xlist = new LinkedList<Integer>();
    ylist = new LinkedList<Integer>();
    //createArray();
  }

  public void setMasterData(int[] md) {
    masterData = md;
  }

  public void setParams(int r, int thresh) {
    radius = r;
    threshold = thresh;
  }

  public int[][] getRegionalMax() {
    return regionalMax;
  }

  public LinkedList<Point> getPossibilityList() {
    return possibilityList;
  }

  public BufferedImage accumulatorImage() {
    BufferedImage toRet = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
    Graphics g = toRet.getGraphics();
    int c = 0;
    int max = 0, mi = 0, mj = 0;
    for (int j = 0; j < accumulator.length; j++) {
      for (int i = 0; i < accumulator[0].length; i++) {
        c = accumulator[j][i];

        if (c == 0)
          g.setColor(Color.black);
        else {
          c = Math.min(c * 8, 255);
          g.setColor(new Color(c, c, c));
          if (regionalMax[j][i] == 1) {
            //g.setColor(Color.white);
            //g.fillOval(i-2, j-2, 4, 4);
            //possibilityList.add(new Point(i,j));
          }
        }
        //g.setColor(new Color(c,c,c));
        g.fillRect(i, j, 1, 1);
        //System.out.print(c+",");
      }
    }
    return toRet;
  }

  public void setSourceImage(BufferedImage img, int[] reg) {
```

```
    sourceImage = img;
    raster = sourceImage.getData();
    accumulator = new int[img.getHeight()][img.getWidth()];
    possibilityList = new LinkedList<Point>();
    valList = new LinkedList<Integer>();
    height = img.getHeight();
    width = img.getWidth();
    region = reg;

}

public void setRegion(int[] reg) {
    region = reg;
}

public void createSourceArray(int[] data, int w, int h) {
    xlist = new LinkedList<Integer>();
    ylist = new LinkedList<Integer>();
    height = h;
    width = w;
    sourceArray = new int[h][w];
    zeroAccumulator = new int[h][w];
    accumulator = new int[h][w];
    for (int j = 0; j < h; j++) {
        for (int i = 0; i < w; i++) {
            sourceArray[j][i] = (data[j * w + i] == -1) ? 255 : 0;
            if (sourceArray[j][i] == 255) {
                if (j >= region[1] && i >= region[0] && i < (region[2] + region[0] - 1) && j < (region[3] + region[1] - 1)) {
                    sourceArray[j][i] = 1;
                    xlist.add(i);
                    ylist.add(j);
                }
            }
            zeroAccumulator[j][i] = 0;
            accumulator[j][i] = 0;
        }
    }
}

public void resetChangeableArrays(int[] reg) {
    //xlist = new LinkedList<Integer>();
    //ylist = new LinkedList<Integer>();
    valList = new LinkedList<Integer>();
    possibilityList = new LinkedList<Point>();
    for (int j = 0; j < accumulator.length; j++) Arrays.fill(accumulator[j], 0);
    //accumulator = zeroAccumulator;
    region = reg;
}

public void setSourceImage_fast(int[] data, int[] reg, int w, int h) {
    xlist = new LinkedList<Integer>();
    ylist = new LinkedList<Integer>();
    valList = new LinkedList<Integer>();
    possibilityList = new LinkedList<Point>();
    accumulator = new int[h][w];
    width = w;
    height = h;
    sourceArray = new int[h][w];
    for (int j = 0; j < h; j++) {
        for (int i = 0; i < w; i++) {
            sourceArray[j][i] = (data[j * w + i] != 0) ? 255 : 0;
            if (sourceArray[j][i] == 255) {
                sourceArray[j][i] = 1;
                xlist.add(i);
                ylist.add(j);
            }
```

```
            accumulator[j][i] = 0;

        }
    }
    //radius = r;
    //threshold = thresh;
    region = reg;
    //xlist = new LinkedList<Integer>();
    //ylist = new LinkedList<Integer>();
}

public void createArray() {
    xlist = new LinkedList<Integer>();
    ylist = new LinkedList<Integer>();
    int w = sourceImage.getWidth();
    int h = sourceImage.getHeight();
    sourceArray = new int[h][w];
    int[] pixels = new int[h * w];
    int[] dummy = new int[10];
    for (int j = 0; j < h; j++) {
        for (int i = 0; i < w; i++) {
            sourceArray[j][i] = raster.getPixel(i, j, dummy)[0];
            if (sourceArray[j][i] == 255) {
                sourceArray[j][i] = 1;
                xlist.add(i);
                ylist.add(j);
            }
            accumulator[j][i] = 0;
        }
    }
}

public void process(boolean b) {
    if (b) createArray();
    int xmin = region[0];
    int xmax = xmin + region[2] - 1;
    int ymin = region[1];
    int ymax = ymin + region[3] - 1;
    //System.out.println(xmin+","+ymin+";"+xmax+","+ymax);
    int xmx, xpx, ypx, ymx;
    int ypy, ymy, xpy, xmy;
    int xCenter, yCenter;
    int x, y;
    double d;
    boolean ypxin, ymxin, xpxin, xmxin;
    boolean ypyin, ymyin, xpyin, xmyin;
    //System.out.println("xlist: " + xlist.size());
    for (int count = 0; count < xlist.size(); count++) {
        xCenter = xlist.get(count);
        yCenter = ylist.get(count);

        xmx = xCenter - radius;
        xpx = xCenter + radius;
        ypx = yCenter + radius;
        ymx = yCenter - radius;

        if (xpx < xmin || xmx > xmax || ypx < ymin || ymx > ymax) {

            break;
        }
        if (ypx < height && ymx >= 0 && xpx < width && xmx >= 0) {
            x = 1;
            y = radius;
            d = 5 / 4 - radius;
            accumulator[ypx][xCenter] = accumulator[ypx][xCenter] + 1;
            accumulator[ymx][xCenter] = accumulator[ymx][xCenter] + 1;
```

```
    accumulator[yCenter][xpx] = accumulator[yCenter][xpx] + 1;
    accumulator[yCenter][xmx] = accumulator[yCenter][xmx] + 1;
    while (y > x) {
      xpx = xCenter + x;
      xmx = xCenter - x;
      ypy = yCenter + y;
      ymy = yCenter - y;
      ypx = yCenter + x;
      ymx = yCenter - x;
      xpy = xCenter + y;
      xmy = xCenter - y;
      accumulator[ypy][xpx] = accumulator[ypy][xpx] + 1;
      accumulator[ymy][xpx] = accumulator[ymy][xpx] + 1;
      accumulator[ypy][xmx] = accumulator[ypy][xmx] + 1;
      accumulator[ymy][xmx] = accumulator[ymy][xmx] + 1;
      accumulator[ypx][xpy] = accumulator[ypx][xpy] + 1;
      accumulator[ymx][xpy] = accumulator[ymx][xpy] + 1;
      accumulator[ypx][xmy] = accumulator[ypx][xmy] + 1;
      accumulator[ymx][xmy] = accumulator[ymx][xmy] + 1;
      if (d < 0) {
        d = d + x * 2 + 3;
        x = x + 1;
      } else {
        d = d + (x - y) * 2 + 5;
        x = x + 1;
        y = y - 1;
      }
    }
    if (x == y) {
      xpx = xCenter + x;
      xmx = xCenter - x;
      ypy = yCenter + y;
      ymy = yCenter - y;
      accumulator[ypy][xpx] = accumulator[ypy][xpx] + 1;
      accumulator[ymy][xpx] = accumulator[ymy][xpx] + 1;
      accumulator[ypy][xmx] = accumulator[ypy][xmx] + 1;
      accumulator[ymy][xmx] = accumulator[ymy][xmx] + 1;
    }
  } else {
    ypxin = ypx >= ymin && ypx <= ymax;
    ymxin = ymx >= ymin && ymx <= ymax;
    xpxin = xpx >= xmin && xpx <= xmax;
    xmxin = xmx >= xmin && xmx <= xmax;

    if (ypxin) accumulator[ypx][xCenter] = accumulator[ypx][xCenter] + 1;
    if (ymxin) accumulator[ymx][xCenter] = accumulator[ymx][xCenter] + 1;
    if (xpxin) accumulator[yCenter][xpx] = accumulator[yCenter][xpx] + 1;
    if (xmxin) accumulator[yCenter][xmx] = accumulator[yCenter][xmx] + 1;
    x = 1;
    y = radius;
    d = 5 / 4 - radius;
    while (y > x) {
      xpx = xCenter + x;
      xpxin = xpx >= xmin & xpx <= xmax;
      xmx = xCenter - x;
      xmxin = xmx >= xmin & xmx <= xmax;
      ypy = yCenter + y;
      ypyin = ypy >= ymin && ypy <= ymax;
      ymy = yCenter - y;
      ymyin = ymy >= ymin && ymy <= ymax;
      ypx = yCenter + x;
      ypxin = ypx >= ymin && ypx <= ymax;
      ymx = yCenter - x;
      ymxin = ymx >= ymin && ymx <= ymax;
      xpy = xCenter + y;
      xpyin = xpy >= xmin && xpy <= xmax;
```

```
            xmy = xCenter - y;
            xmyin = xmy >= xmin && xmy <= xmax;
            if (ypyin && xpxin) accumulator[ypy][xpx] = accumulator[ypy][xpx] + 1;
            if (ymyin && xpxin) accumulator[ymy][xpx] = accumulator[ymy][xpx] + 1;
            if (ypyin && xmxin) accumulator[ypy][xmx] = accumulator[ypy][xmx] + 1;
            if (ymyin && xmxin) accumulator[ymy][xmx] = accumulator[ymy][xmx] + 1;
            if (ypxin && xpyin) accumulator[ypx][xpy] = accumulator[ypx][xpy] + 1;
            if (ymxin && xpyin) accumulator[ymx][xpy] = accumulator[ymx][xpy] + 1;
            if (ypxin && xmyin) accumulator[ypx][xmy] = accumulator[ypx][xmy] + 1;
            if (ymxin && xmyin) accumulator[ymx][xmy] = accumulator[ymx][xmy] + 1;
            if (d < 0) {
               d = d + x * 2 + 3;
               x = x + 1;
            } else {
               d = d + (x - y) * 2 + 5;
               x = x + 1;
               y = y - 1;
            }

         }
         if (x == y) {
            xpx = xCenter + x;
            xpxin = xpx >= xmin & xpx <= xmax;
            xmx = xCenter - x;
            xmxin = xmx >= xmin & xmx <= xmax;
            ypy = yCenter + y;
            ypyin = ypy >= ymin & ypy <= ymax;
            ymy = yCenter - y;
            ymyin = ymy >= ymin & ymy <= ymax;
            if (ypyin && xpxin) accumulator[ypy][xpx] = accumulator[ypy][xpx] + 1;
            if (ymyin && xpxin) accumulator[ymy][xpx] = accumulator[ymy][xpx] + 1;
            if (ypyin && xmxin) accumulator[ypy][xmx] = accumulator[ypy][xmx] + 1;
            if (ymyin && xmxin) accumulator[ymy][xmx] = accumulator[ymy][xmx] + 1;
         }
      }
   }
   regionalMax = computeRegionalMax(accumulator, 16, threshold);
}

public LinkedList<Integer> valList;

public int[][] computeRegionalMax(int[][] in, int window, int thresh) {
   int out[][] = new int[in.length][in[0].length];
   for (int j = 0; j < in.length; j += window) {
      for (int i = 0; i < in[0].length; i += window) {
         int regMax = 0;
         for (int y = j; y < j + window; y++) {
            for (int x = i; x < i + window; x++) {
               if (x < in[0].length && x >= 0 && y < in.length && y >= 0) {
                  //System.out.println(x+","+y+" :: " + in[0].length + ", " + in.length);
                  regMax = Math.max(regMax, in[y][x]);
               }
            }
         }
         for (int y = j; y < j + window; y++) {
            for (int x = i; x < i + window; x++) {
               if (x < in[0].length && x >= 0 && y < in.length && y >= 0) {
                  if (in[y][x] == regMax && in[y][x] >= thresh) {
                     out[y][x] = 1;
                     possibilityList.add(new Point(x, y));
                     valList.add(in[y][x]);
                     //System.out.println(in[y][x] + " @ ("+x+","+y+")");
                  } else {
                     out[y][x] = 0;
                  }
               }
```

```
            }

          }
        }
      }
      return out;
    }
}
```