World Journal of Advanced Technology & Engineering
Vol. 1, No. 1, April 2014, pp. 1 - 19
Available online at http://scenrp.com/Journals.php

Research article

Development of Continuous Oscillatory Baffled Reactor Arrangement for Biodiesel Production from Jatropha oil.

B. K. Highina[1], I. M. Bugaje,[2] and G. M. Nglala[3]

[1]Department of Chemical Engineering, University of Maiduguri P.M.B1069, Maiduguri, Borno State, Nigeria
[2]Department of Chemical Engineering, Ahmadu Bello University Zaria, Kaduna State, Nigeria.
[3]Department of Mechanical Engineering, University of Maiduguri, P.M.B1069,Maiduguri
Borno State, Nigeria.
Corresponding Authors; phone: +2348038514244, Email: bkhighina@gmail.com

ABSTRACT

This research work involved the design and development of continuous oscillatory baffled reactor arrangement for biodiesel production from Jatropha oil using software. The software was developed using Visual Basic Programming Language for the preliminary design of the reactor. Some of the reactor geometrical configurations obtained from the software include; internal diameter, length and volume of the reactor which are 0.1 m, 3.1 m and $2.4x10^{-3}$ m^3 respectively. Also, the volumetric flow rate and the total energy required for reactor operation were $1.34x10^{-5}$ m/s and $3.126x10^{-3}$ W/m respectively. Moreover, the total number of tanks in series in the reactor was determined to be 19. Hence, the reactor was constructed taking into consideration the engineering material selections after its detailed design. The performance evaluation of the COBRA was carried out theoretically by comparing the Power density against length to diameter ratio for COBRA and Plug flow reactor. . It was clearly seen that length-to-diameter ratio is much smaller for the continuous oscillatory baffled reactor arrangement at similar power densities, meaning that, an oscillatory flow reactor will be shorter, typically by at least fifty times of magnitude than plug flow reactor. The dependence of the residence time distribution performance on the velocity ratio of COBRA was also studied. The velocity ratio increases from 1.0 to 8.0 as the number of tanks in series decreases from 85.0 to 10.0. Also, above the velocity ratio of 8.0 the number of tanks in series drops below 10.0. This result confirms that a better mixing in a continuous oscillatory baffled reactor is obtained when a velocity ratio is in the range of 2.0 to 8.0 while the number of tanks in series is above 10.0.

Keywords: Development, COBRA, Software, Biodiesel, Jatropha oil

INTRODUCTION

The Continuous Oscillatory Baffled Reactor Arrangement (COBRA) is a relatively new design of a continuous reactor consisting of tubes containing equally spaced orifice plate baffles [1]. This reactor design provides solutions to problems due to batch and conventional continuous reactors. It is a plug flow reactor in which the achievement of good plug is decoupled from the net flow. In practice, this means that it can accommodate long reactions in a reactor of L/D ratio order of magnitude less than conventional plug reactors. It has been recognized that the batch stirred reactor is a primary mode used in the synthesis of biodiesel. However, plusatile flow has been extensively researched and the fundamental principles have been successfully developed upon which its hydrodynamics are based. Oscillatory flow biodiesel reactor offers precise control of mixing by means of the baffle geometry and pulsation which facilitates continuous operation giving plug flow residence time distribution with high turbulence and enhanced mass and heat transfer [2]. Oscillatory flow in baffled tubes has been studied for many years and much work has been done in areas related to fluid dynamics, heat and mass transfer and residence time distribution. Many advantages have been characterized for oscillatory flow mixing, such as efficient dispersion for immiscible fluid, uniform particles suspension, gas-in-liquid dispersion and multiphase mixing [4,7].

Continuous processing is achieved by super-imposing an oscillatory flow on a steady throughout in a tubular reactor. Each inter-baffled zone is a unit stirred tank, and in a multi-pass configuration, a multiplicity of stirred tank exists in series, allowing a close approach to plug RTD, even at low throughput rates [8].

Mackley and Brunold *et al*. [4,3] observed that oscillatory flows through geometries in which a surface was presented transversely to the flow could produce uniform and efficient mixing patterns. The optimal geometry for the uniform mixing of fluid within a tube was found to be equally spared orifice plates 1.5 tubes diameter apart of 0.25 fractional open cross sectional area.

EXPERIMENTAL METHODS

The geometric configuration was a proposed approach to design oscillatory flow biodiesel reactor (COBRA) by maintaining the dynamic similarity using various dimensionless groups, followed by the application of other empirical design correlations specific to oscillatory flows [2,8]. Scaling up oscillatory flow reactor can be assumed by keeping two geometric ratios constant such as the baffle spacing (L) which is expressed as a fixed ratio of the tube diameter, and the baffle orifice open area (S) as stated in the following equations [8,9]. The dimensionless groups used to characterize the COBRA behaviour are as follows:

Net flow Reynolds number, (Re_n): $$Re_n = \frac{\rho u D}{\mu} \quad (1)$$

Oscillatory Reynolds number, (Re_o) $$Re_o = \frac{\rho 2\pi f x_o D}{\mu} \quad (2)$$

Velocity ratio, (ψ) $$\psi = \frac{Re_o}{Re_n} \quad (3)$$

Strouhal number, (St_r) $$St_r = \frac{D}{4\pi x_o} \quad (4)$$

Where ρ is the fluid density, u, the net flow velocity, D, the tube diameter, μ the viscosity, *f*, the oscillatory frequency and X_o, the centre- to -peak amplitude. It was shown that if a velocity ratio of between approximately 4 and 10 were maintained, then a good approximation to plug flow would be achieved. This can be achieved along with heat transfer and mass transfer enhancement [8, 5].

$$L=1.5D \quad (5)$$

$$S= d^2_0/D^2 \quad (6)$$

Stonestreet and Harvey reported that the fractional open area of baffle is in the range of 0.2 – 0.4 usually 0.25, such that the orifice diameter is half the tube diameter [8].

The key dimensionless numbers that must be kept in a similar range on the full scale, as on the pilot or laboratory scale are equations 1, 2 and 4.
Based on the selection of net Reynolds number (Re_n) and oscillatory Reynolds number (Re_o) values and the required residence time, the superficial velocity (U), the reactor length (Z), volume (V) and the volumetric flow rate (Q) were determined as follows;

$$U=\frac{\mu \mathrm{Re}_n}{D\rho} \quad (7)$$

$$Z=Ut \quad (8)$$

$$V=\frac{Z\pi D^2}{4} \quad (9)$$

$$Q=\frac{V}{t} \quad (10)$$

The pressure drop due to net flow through a baffle tube can be obtained by the standard equation for flow through an orifice modified to account for the total number of identical baffles (M_i) [3].

$$\Delta P = \frac{m_i \rho u^2}{2C_o\left(\frac{1}{S^2}-1\right)} \quad (11)$$

Overall power per unit volume or power density for net flow and oscillatory flow through baffled tube can be calculated using the density of fluid (ρ), angular frequency of oscillatory (ω), amplitude of oscillation (χ_o) and eddy mixing length (l) in the following equations:

$$\varepsilon_n = \frac{\Delta P a_c}{Vu} = \frac{\Delta P u}{Z\varepsilon} \quad (12)$$

$$\varepsilon_v = \frac{3m_i \rho \omega^3 \chi_o^3 l}{Sz} \quad (13)$$

Development of Software for Design of COBRA

The software was developed by adopting the mixing –based methodology given by Stonestreet and Harvey [8] for continuous oscillatory flow reactor. The software was used for the preliminary design of the reactor.
In developing the software, a Visual Basic programming language was used.

PROGRAMMING CODES FOR DESIGN SOFTWARE OF COBRA

```
System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.ComponentModel;
using Ark.Math;

namespace COBRWpfApplication using
{
   public enum DesignStatus { Viable, VolumetricFlowRateFail, TankInSeriesNumberFail, TotalPowerFail };

   [Serializable()]
   public class OFR : INotifyPropertyChanged
   {
      [field:NonSerialized()]
      public event PropertyChangedEventHandler PropertyChanged;
```

```
private void NotifyPropertyChanged(String propertyName)
{
   if (PropertyChanged != null)
   {
      PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
   }
}

private double _MeanResidenceTime;
public double MeanResidenceTime
{
   get
   {
      return _MeanResidenceTime;
   }
   set
   {
      if (_MeanResidenceTime != value)
      {
         _MeanResidenceTime = value;
         NotifyPropertyChanged("MeanResidenceTime");
      }
   }
}

private double _ReynoldsNumberOscillation;
public double ReynoldsNumberOscillation
{
   get
   {
      return _ReynoldsNumberOscillation;
   }
   set
   {
      if (_ReynoldsNumberOscillation != value)
      {
         _ReynoldsNumberOscillation = value;
         NotifyPropertyChanged("ReynoldsNumberOscillation");
      }
   }
}

private double _ReynoldsNumberNetFlow;
public double ReynoldsNumberNetFlow
{
   get
   {
      return _ReynoldsNumberNetFlow;
   }
   set
   {
      if (_ReynoldsNumberNetFlow != value)
      {
         _ReynoldsNumberNetFlow = value;
         NotifyPropertyChanged("ReynoldsNumberNetFlow");
      }
   }
}
```

```
private double _StrouhalNumber;
public double StrouhalNumber
{
   get
   {
      return _StrouhalNumber;
   }
   set
   {
      if (_StrouhalNumber != value)
      {
         _StrouhalNumber = value;
         NotifyPropertyChanged("StrouhalNumber");
      }
   }
}


private double _StandardOrificeCoefficient;
public double StandardOrificeCoefficient
{
   get
   {
      return _StandardOrificeCoefficient;
   }
   set
   {
      if (_StandardOrificeCoefficient != value)
      {
         _StandardOrificeCoefficient = value;
         NotifyPropertyChanged("StandardOrificeCoefficient");
      }
   }
}

private double _DesignVolumetricFlowRate;
public double DesignVolumetricFlowRate
{
   get
   {
      return _DesignVolumetricFlowRate;
   }
   set
   {
      if (_DesignVolumetricFlowRate != value)
      {
         _DesignVolumetricFlowRate = value;
         NotifyPropertyChanged("DesignVolumetricFlowRate");
      }
   }
}

private double _FluidViscosity;
public double FluidViscosity
{
   get
   {
      return _FluidViscosity;
   }
```

```
    set
    {
      if (_FluidViscosity != value)
      {
        _FluidViscosity = value;
        NotifyPropertyChanged("FluidViscosity");
      }
    }
}

private double _FluidDensity;
public double FluidDensity
{
    get
    { Total Energy ,E_t,(kJ)
      return _FluidDensity;
    }
    set
    {
      if (_FluidDensity != value)
      {
        _FluidDensity = value;
        NotifyPropertyChanged("FluidDensity");
      }
    }
}

private double _VelocityRatio;
public double VelocityRatio
{
    get
    {
      return _VelocityRatio;
    }
    set
    {
      if (_VelocityRatio != value)
      {
        _VelocityRatio = value;
        NotifyPropertyChanged("VelocityRatio");
      }
    }
}
public void ComputeVelocityRatio()
{
    VelocityRatio = ReynoldsNumberOscillation / ReynoldsNumberNetFlow;
}

ObservableCollection<TubeInternalDiameter> _TubeInternalDiameters;
public ObservableCollection<TubeInternalDiameter> TubeInternalDiameters
{
    get
    {
      if (_TubeInternalDiameters == null)
        _TubeInternalDiameters = new ObservableCollection<TubeInternalDiameter>();
      return _TubeInternalDiameters;
    }
    set
    {
```

```
            if (_TubeInternalDiameters != value)
            {
               _TubeInternalDiameters = value;
               NotifyPropertyChanged("TubeInternalDiameters");
            }
         }
      }

      ObservableCollection<DesignSimulation> _DesignSimulations;
      public ObservableCollection<DesignSimulation> DesignSimulations
      {
         get
         {
            return _DesignSimulations;
         }
         set
         {
            if (_DesignSimulations != value)
            {
               _DesignSimulations = value;
               NotifyPropertyChanged("DesignSimulations");
            }
         }
      }

      public void Simulate()
      {
         ObservableCollection<DesignSimulation> items = new ObservableCollection<DesignSimulation>();
         ComputeVelocityRatio();
         TubeInternalDiameters = new
ObservableCollection<TubeInternalDiameter>(TubeInternalDiameters.OrderBy(t => t.Value));

         foreach(var t in TubeInternalDiameters)
         {
            DesignSimulation item = new DesignSimulation();
            DesignStatus ds = item.Simulate(t.Value, ReynoldsNumberOscillation, ReynoldsNumberNetFlow,
VelocityRatio, StrouhalNumber, MeanResidenceTime, DesignVolumetricFlowRate, FluidDensity,
FluidViscosity, StandardOrificeCoefficient);

            //if (ds == DesignStatus.VolumetricFlowRateFail)
            //    break;

            //if (ds == DesignStatus.TankInSeriesNumberFail)
            //    continue;

            items.Add(item);
         }

         DesignSimulations = items;
      }

   }

   [Serializable()]
   public class TubeInternalDiameter : INotifyPropertyChanged
   {
      [field:NonSerialized()]
      public event PropertyChangedEventHandler PropertyChanged;
      private void NotifyPropertyChanged(String propertyName)
```

```
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
        }

        private double _Value;
        public double Value
        {
            get
            {
                return _Value;
            }
            set
            {
                if (_Value != value)
                {
                    _Value = value;
                    NotifyPropertyChanged("Value");
                }
            }
        }2
    }

    [Serializable()]
    public class DesignSimulation : INotifyPropertyChanged
    {
        [field:NonSerialized()]
        public event PropertyChangedEventHandler PropertyChanged;
        private void NotifyPropertyChanged(String propertyName)
        {
            if (PropertyChanged != null)
            {
                PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
            }
        }

        public DesignStatus Simulate(double tubeInternalDiameter, double reynoldsNumberOscillation, double
reynoldsNumberNetFlow, double velocityRatio, double strouhalNumber, double meanResidenceTime, double
designVolumetricFlowRate, double fluidDensity, double fluidViscosity, double standardOrificeCoefficient)
        {
            TubeInternalDiameter = tubeInternalDiameter;

            ComputeSuperficialTubeVelocity(reynoldsNumberNetFlow, fluidDensity, fluidViscosity);
            ComputeTotalTubeLength(meanResidenceTime);
            ComputeTubeVolume();
            ComputeVolumetricFlowRate(meanResidenceTime);

            //if (VolumetricFlowRate != designVolumetricFlowRate)
            //    return DesignStatus.VolumetricFlowRateFail;

            ComputeBaffleSpacing();
            ComputeTotalBaffleNumber();
            ComputeTankInSeriesNumber();

            //if (TankInSeriesNumber <= 10)
            //    return DesignStatus.TankInSeriesNumberFail;
```

```
            ComputeBaffleOrificeDiameter();
            ComputeBaffleFractionalOpenArea();
            ComputeAmplitude(strouhalNumber);
            ComputeAngularFrequency(reynoldsNumberOscillation, fluidDensity, fluidViscosity);
            ComputeEnergyNetFlow(velocityRatio, fluidDensity, standardOrificeCoefficient);
            ComputeAverageEnergyPerVolume(fluidDensity, standardOrificeCoefficient);
            ComputeEddyAverageEnergyPerVolume(fluidDensity, standardOrificeCoefficient);
            ComputeTotalEnergy();

            return DesignStatus.Viable;
        }

        private double _TubeInternalDiameter;
        public double TubeInternalDiameter
        {
            get
            {
                return _TubeInternalDiameter;
            }
            set
            {
                if (_TubeInternalDiameter != value)
                {
                    _TubeInternalDiameter = value;
                    NotifyPropertyChanged("TubeInternalDiameter");
                }
            }
        }

        private double _SuperficialTubeVelocity;
        public double SuperficialTubeVelocity
        {
            get
            {
                return _SuperficialTubeVelocity;
            }
            set
            {
                if (_SuperficialTubeVelocity != value)
                {
                    _SuperficialTubeVelocity = value;
                    NotifyPropertyChanged("SuperficialTubeVelocity");
                }
            }
        }
        public void ComputeSuperficialTubeVelocity(double reynoldsNumberNetFlow, double fluidDensity,
double fluidViscosity)
        {
            SuperficialTubeVelocity = fluidViscosity * reynoldsNumberNetFlow / fluidDensity /
TubeInternalDiameter;
        }

        private double _TotalTubeLength;
        public double TotalTubeLength
        {
            get
            {
                return _TotalTubeLength;
            }
```

```
    set
    {
      if (_TotalTubeLength != value)
      {
        _TotalTubeLength = value;
        NotifyPropertyChanged("TotalTubeLength");
      }
    }
}
public void ComputeTotalTubeLength(double meanResidenceTime)
{
    TotalTubeLength = SuperficialTubeVelocity * meanResidenceTime;
}

private double _TubeVolume;
public double TubeVolume
{
    get
    {
      return _TubeVolume;
    }
    set
    {
      if (_TubeVolume != value)
      {
        _TubeVolume = value;
        NotifyPropertyChanged("TubeVolume");
      }
    }
}
public void ComputeTubeVolume()
{
    TubeVolume = TotalTubeLength * Math.PI * Math.Pow(TubeInternalDiameter, 2.0) / 4.0;
}

private double _VolumetricFlowRate;
public double VolumetricFlowRate
{
    get
    {
      return _VolumetricFlowRate;
    }
    set
    {
      if (_VolumetricFlowRate != value)
      {
        _VolumetricFlowRate = value;
        NotifyPropertyChanged("VolumetricFlowRate");
      }
    }
}
public void ComputeVolumetricFlowRate(double meanResidenceTime)
{
    VolumetricFlowRate = TubeVolume / meanResidenceTime;
}

private double _BaffleOrificeDiameter;
public double BaffleOrificeDiameter
{
```

```
    get
    {
      return _BaffleOrificeDiameter;
    }
    set
    {
      if (_BaffleOrificeDiameter != value)
      {
        _BaffleOrificeDiameter = value;
        NotifyPropertyChanged("BaffleOrificeDiameter");
      }
    }
  }
  public void ComputeBaffleOrificeDiameter()
  {
    BaffleOrificeDiameter = TubeInternalDiameter / 2.0;
  }

  private double _BaffleSpacing;
  public double BaffleSpacing
  {
    get
    {
      return _BaffleSpacing;
    }
    set
    {
      if (_BaffleSpacing != value)
      {
        _BaffleSpacing = value;
        NotifyPropertyChanged("BaffleSpacing");
      }
    }
  }
  public void ComputeBaffleSpacing()
  {
    BaffleSpacing = TubeInternalDiameter * 1.5;
  }

  private double _TotalBaffleNumber;
  public double TotalBaffleNumber
  {
    get
    {
      return _TotalBaffleNumber;
    }
    set
    {
      if (_TotalBaffleNumber != value)
      {
        _TotalBaffleNumber = value;
        NotifyPropertyChanged("TotalBaffleNumber");
      }
    }
  }
  public void ComputeTotalBaffleNumber()
  {
    TotalBaffleNumber = TotalTubeLength / BaffleSpacing;
  }
```

```
        private double _TankInSeriesNumber;
        public double TankInSeriesNumber
        {
            get
            {
                return _TankInSeriesNumber;
            }
            set
            {
                if (_TankInSeriesNumber != value)
                {
                    _TankInSeriesNumber = value;
                    NotifyPropertyChanged("TankInSeriesNumber");
                }
            }
        }
        public void ComputeTankInSeriesNumber()
        {
            TankInSeriesNumber = TotalBaffleNumber - 1.0;
        }

        private double _BaffleFractionalOpenArea;
        public double BaffleFractionalOpenArea
        {
            get
            {
                return _BaffleFractionalOpenArea;
            }
            set
            {
                if (_BaffleFractionalOpenArea != value)
                {
                    _BaffleFractionalOpenArea = value;
                    NotifyPropertyChanged("BaffleFractionalOpenArea");
                }
            }
        }
        public void ComputeBaffleFractionalOpenArea()
        {
            BaffleFractionalOpenArea = Math.Pow(BaffleOrificeDiameter, 2.0) / Math.Pow(TubeInternalDiameter,
2.0);
        }

        private double _Amplitude;
        public double Amplitude
        {
            get
            {
                return _Amplitude;
            }
            set
            {
                if (_Amplitude != value)
                {
                    _Amplitude = value;
                    NotifyPropertyChanged("Amplitude");
                }
            }
```

```
    }
    public void ComputeAmplitude(double strouhalNumber)
    {
      Amplitude = TubeInternalDiameter / strouhalNumber / Math.PI / 4.0;
    }

    private double _AngularFrequency;
    public double AngularFrequency
    {
      get
      {
        return _AngularFrequency;
      }
      set
      {
        if (_AngularFrequency != value)
        {
          _AngularFrequency = value;
          NotifyPropertyChanged("AngularFrequency");
        }
      }
    }
    public void ComputeAngularFrequency(double reynoldsNumberOscillation, double fluidDensity, double
fluidViscosity)
    {
      AngularFrequency = reynoldsNumberOscillation * fluidViscosity / Amplitude / TubeInternalDiameter /
fluidDensity;
    }

    private double _EnergyNetFlow;
    public double EnergyNetFlow
    {
      get
      {
        return _EnergyNetFlow;
      }
      set
      {
        if (_EnergyNetFlow != value)
        {
          _EnergyNetFlow = value;
          NotifyPropertyChanged("EnergyNetFlow");
        }
      }
    }
    public void ComputeEnergyNetFlow(double velocityRatio, double fluidDensity, double
standardOrificeCoefficient)
    {
      EnergyNetFlow = TotalBaffleNumber * fluidDensity * Math.Pow(SuperficialTubeVelocity, 2.0) / 2.0 /
standardOrificeCoefficient * (Math.Pow(BaffleFractionalOpenArea, 2.0).Reciprocal() - 1.0) *
(SuperficialTubeVelocity / TotalTubeLength) * (1 + Math.Pow((4.0 * velocityRatio / Math.PI), 3.0)).Root(3);
    }

    private double _AverageEnergyPerVolume;
    public double AverageEnergyPerVolume
    {
      get
      {
        return _AverageEnergyPerVolume;
```

```
        }
        set
        {
            if (_AverageEnergyPerVolume != value)
            {
                _AverageEnergyPerVolume = value;
                NotifyPropertyChanged("AverageEnergyPerVolume");
            }
        }
    }
    public void ComputeAverageEnergyPerVolume(double fluidDensity, double standardOrificeCoefficient)
    {
        AverageEnergyPerVolume = 2.0 * TotalBaffleNumber * fluidDensity * Math.Pow((Amplitude * AngularFrequency), 3.0) / 3.0 / Math.PI / Math.Pow(standardOrificeCoefficient, 2.0) / TotalTubeLength * (Math.Pow(BaffleFractionalOpenArea, 2.0).Reciprocal() - 1.0);
    }

    private double _EddyAverageEnergyPerVolume;
    public double EddyAverageEnergyPerVolume
    {
        get
        {
            return _EddyAverageEnergyPerVolume;
        }
        set
        {
            if (_EddyAverageEnergyPerVolume != value)
            {
                _EddyAverageEnergyPerVolume = value;
                NotifyPropertyChanged("EddyAverageEnergyPerVolume");
            }
        }
    }
    public void ComputeEddyAverageEnergyPerVolume(double fluidDensity, double standardOrificeCoefficient)
    {
        EddyAverageEnergyPerVolume = 3.0 * TotalBaffleNumber * fluidDensity * Math.Pow(AngularFrequency, 3.0) * Math.Pow(Amplitude, 2.0) * 0.009 / BaffleFractionalOpenArea / TotalTubeLength;
    }

    private double _TotalEnergy;
    public double TotalEnergy
    {
        get
        {
            return _TotalEnergy;
        }
        set
        {
            if (_TotalEnergy != value)
            {
                _TotalEnergy = value;
                NotifyPropertyChanged("TotalEnergy");
            }
        }
    }
    public void ComputeTotalEnergy()
    {
```

```
        TotalEnergy = EnergyNetFlow + AverageEnergyPerVolume;
    }
}
```

RESULTS AND DISCUSSIONS

SIMULATED RESULTS FOR THE GEOMETRICAL CONFIGURATION OF COBRA

Tube internal diameter D(m)	Superficial velocity U (m/s)	Length of reactor tube Z(m)	Volume of reactor V(m^3)	Volumetric flow rate Vo (m^3/s)	Baffle spacing L(m)	Number of Baffles, M	Number of tanks in series N	Baffle orifice diameter d_o(m)
0.025	0.006827	12.28884	0.006032	3.35E-06	0.0375	327.7024	326.7024	0.0125
0.05	0.003414	6.14442	0.012065	6.70E-06	0.075	81.9256	80.9256	0.025
0.1	0.001707	3.07221	0.024129	1.34E-05	0.15	20.4814	19.4814	0.05
0.2	0.000853	1.536105	0.048258	2.68E-05	0.3	5.12035	4.12035	0.1
0.3	0.000569	1.02407	0.072387	4.02E-05	0.45	2.275711	1.275711	0.15
0.4	0.000427	0.768053	0.096516	5.36E-05	0.6	1.280088	0.280088	0.2
0.5	0.000341	0.614442	0.120645	6.70E-04	0.75	0.819256	-0.18074	0.25
0.6	0.000284	0.512035	0.144774	8.04E-05	0.9	0.568928	-0.43107	0.3
0.7	0.000244	0.438887	0.168904	9.38E-05	1.05	0.417988	-0.58201	0.35
0.8	0.000213	0.384026	0.193033	0.000107	1.2	0.320022	-0.67998	0.4
0.9	0.00019	0.341357	0.217162	0.000121	1.35	0.252857	-0.74714	0.45
1.0	0.000171	0.307221	0.241291	0.000134	1.5	0.204814	-0.79519	0.5

Tube internal diameter D, (m)	Amplitude X(m)	Frequency (radians/s)	Energy net flow $E_{n,}$(W/m)	Average Energy $E_{v,}$(W/m)	Eddy average energy (W/m)	Total Energy $E_{t,}$(W/m)
0.025	0.009947	1.372677	0.251763	0.548615	0.673665	0.800378
0.05	0.019894	0.343169	0.015735	0.034288	0.021052	0.050024
0.1	0.039789	0.085792	0.000983	0.002143	0.000658	0.003126
0.2	0.079577	0.021448	6.15E-05	0.000134	2.06E-05	0.000195
0.3	0.119366	0.009532	1.21E-05	2.65E-05	2.71E-06	3.86E-05
0.4	0.159155	0.005362	3.84E-06	8.37E-06	6.42E-07	1.22E-05
0.5	0.198944	0.003432	1.57E-06	3.43E-06	2.11E-07	5.00E-06
0.6	0.238732	0.002383	7.59E-07	1.65E-06	8.46E-08	2.41E-06
0.7	0.278521	0.001751	4.10E-07	8.93E-07	3.91E-08	1.30E-06

0.8	0.31831	0.001341	2.40E-07	5.23E-07	2.01E-08	7.63E-07
0.9	0.358099	0.001059	1.50E-07	3.27E-07	1.11E-08	4.77E-07
1.0	0.397887	0.000858	9.83E-08	2.14E-07	6.58E-09	3.13E-07

Figure1: Constructed COBRA

Theoretical studies on COBRA

The performance evaluation of the COBRA was carried out by comparing the Power density against length to diameter ratio for COBRA and Plug flow reactor. The theoretical study was carried out to ascertain whether or not the chosen configuration requires large amount of energy to achieve the required mixing and flow rate. A useful way of assessing the configurations was to plot the power density against the length to diameter ratio. A graphical comparison is shown in Figure 2 and Figure 3 for continuous oscillatory baffled reactor arrangement and plug flow reactor respectively. It can clearly be seen that length to diameter ratio is much smaller for the continuous oscillatory baffled reactor arrangement at similar power densities, meaning that, an oscillatory flow reactor will be shorter, typically by at least two orders of magnitude than plug flow reactor. This is in accordance with the works of Stonestreet and Harvey [9]. One of the advantages of continuous oscillatory baffled reactor arrangement is the control over the mixing conditions that they offer, and they are therefore likely to be used in applications where achieving the correct degree of mixing is very important. The dependence of the residence time distribution

performance on the velocity ratio can be seen in Figure 4. The velocity ratio increases from 1.0 to 8.0 as the number of tanks in series decreases from 85.0 to 10.0. Also, above the velocity ratio of 8.0 the number of tanks in series drops below 10.0. A better mixing in a continuous oscillatory baffled reactor is obtained when a velocity ratio is in the range of 2.0 to 6.0 and the number of tanks in series is above 10.0 [8].

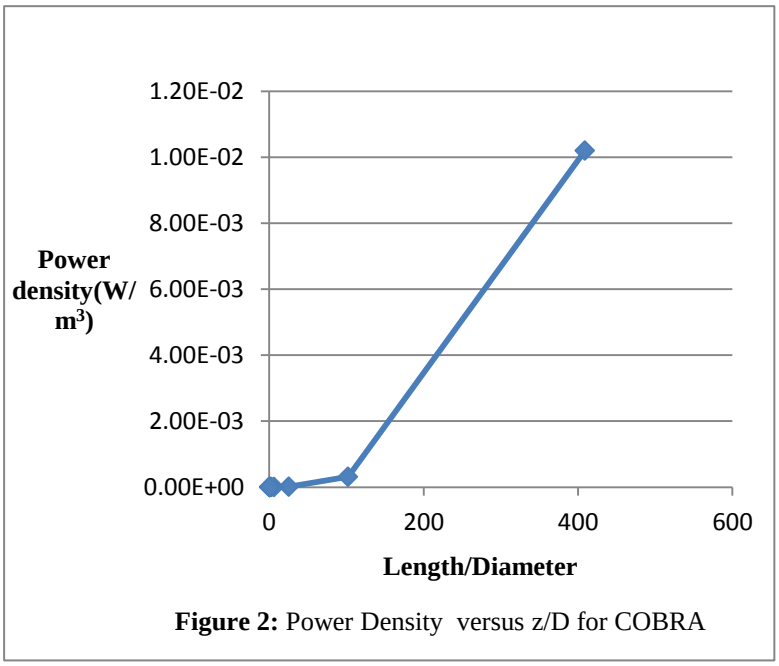

Figure 2: Power Density versus z/D for COBRA

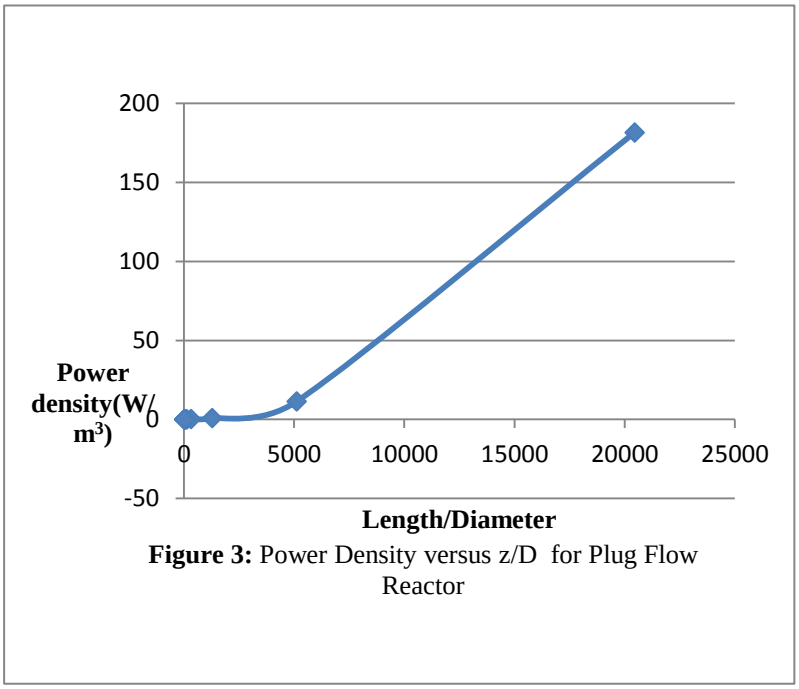

Figure 3: Power Density versus z/D for Plug Flow Reactor

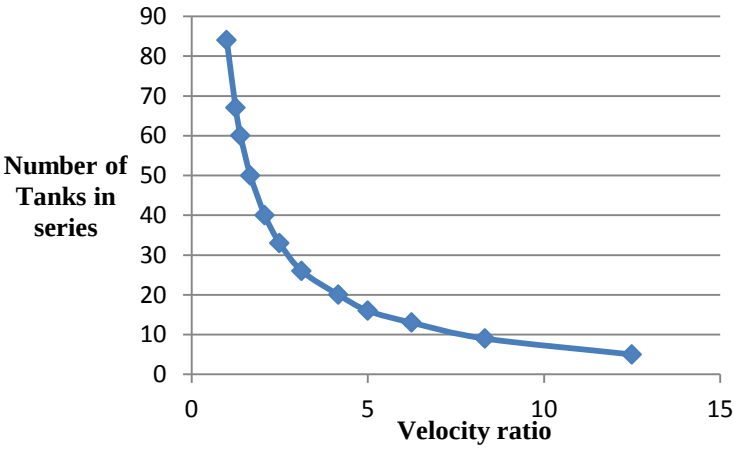

Figure 4: Number of Tanks Versus Velocity Ratio

CONCLUSIONS

Software for the design of the Continuous Oscillatory Baffled Reactor Arrangement was developed and was used in the preliminary design to obtain the geometrical configuration of the reactor. The reactor internal diameter, reactor length and reactor volume were determined to be 0.1m, 3.1m and 0.0024129 m3 respectively. Also, volumetric flow rate and total energy required for the operation of the reactor were found to be 0.0000134 m/s and 0.003126 W/m respectively.

The continuous oscillatory baffled reactor arrangement was fabricated using appropriate engineering materials such as stainless steel, galvanized steel, PVC pipes and glasses.

The theoretical study was carried out on the COBRA. Firstly, the continuous oscillatory baffled reactor arrangement was compared with plug flow reactor theoretically. This was done by comparing the length to diameter ratios of COBRA and plug flow reactor at similar power density. It was found that the length to diameter ratio is much smaller for the COBRA by about 50 times magnitude than the plug flow reactor at similar power density. This means that COBRA will be shorter than the plug flow reactor. Also, theoretical studies on the dependency of the residence time distribution on the velocity ratio were carried out. From the study, it shows that as the velocity ratio increases from 1.0 to 8.0, the number of tanks in series increases. However, above velocity ratio of 8.0 the number of tanks begins to drop below 10.0.

REFERENCES

[1] Adam, P.H., Malconir, R.M. and Thomas, S., (2003). Process intensification of biodiesel production using a continuous oscillatory flow reactor. Journal of Chemical Technology and Biotechnology,78:338-341.

[2] Azhart T.; I. M.Ghazi, M.F.M., Gunam, R.Yunus, T.C.Shean Yaw, (2008). Preliminary design of oscillatory flow biodiesel reactor for continuous biodiesel production from Jatropha Triglycerides.Journal of Engineering Science and Technology 3(2):138-145.

[3] Brunold, C. R., Hunns, J. C. B., Mackley, M. R. And Thompson, J. W., (1989). Experimental observations on flow patterns and energy losses for oscillatory flow in ducts. Journal of Chemical Engineering Science, 20:234-245.

[4] Mackley, M.R., (1991). Process innovation using oscillatory flow within baffled tubes. Trans I Chem E 69: 197-199.

[5] Ni, X., Gao, S., Cumming, R.H. and Pritchard, D.W., (1995). A comparative-study of mass-transfer in yeast for a batch pulsed baffled bioreactor and a stirred-tank fermenter, Chem Eng Sci, 50(13): 2127–2136.

[6] Ni, X., Mackley, M.R., Harvey, A.P., Stonestreet, P., Baird, M.H.I. and Rama Rao, N.V., (2003). Mixing through oscillations and pulsations—a guide to achieving process enhancements in the chemical and process industries, Trans IChemE Part A, 81: 373–383.

[7] Roberts, E.P.L. and Mackley, M.K., (1995). The Simulation of Stretch rates for the quantitative production and mapping of mixing within a channel flow. Chem. Eng. Science, 50 (23): 3727-3746.

[8] Stonestreet, P. and Harvey, A.P., (2002). A mixing-based design methodology for continuous oscillatory flow reactor. Institution of Chemical Engineers, 80: 31-44.

[9] Stonestreet, P. and Mackley, M.R., (2002). Evaluation of Oscillatory flow mixing in tubular reactor for continuous processing. Process intensification in practice (BHR Group Conferences), 247-3746.